Code Composer Studio(CCS)
集成开发环境（IDE）入门指导书

【美】Texas Instruments Incorprated 著

牛金海 等编译

前 言

关于本书

本书根据 Texas Instrument（TI）的 datasheet SPRU509F.pdf 编译而成。

为开始使用Code Composer Studio IDE™，建议首先仔细阅读本书的头两节。其余各节载有对特定的程序和工具更详细的资料。要确定是否可以使用这些功能，可以查看Code Composer Studio IDE提供的在线帮助。

本书有以下特色：

1、关键词中英文对照互译，可以更好地体现 TI datasheet 的原意

2、增加了编译者在项目开发过程中的经验知识

3、对重要的知识点做了强调与诠释

内容简介

本书主要介绍 CCS 开发环境的使用。适合于从事 TI DSP 开发的工程技术人员以及高校的学生参考。

序

经过一个学期的努力，CCS 集成开发环境入门手册，终于与大家见面了。在本书的编译过程中，上海交通大学生物医学工程系的车树明同学对全书进行了详细的阅读与校对，参与本书编译的还有上海交通大学生物医学工程系的硕士研究生罗兰，卢善好，蔡任燕，王纬超，封晓瑞，张驰，周业，陈琦，谭黎明，罗汇，俞培春，孙静静，陈佳铭，赵冀，屈兆辉，谭飞，李德辉等同学。在此一并感谢。

TI 大学计划部经理沈洁女士，潘亚涛先生，黄争先生对本书的出版给予很多支持与帮助。

上海交通大学 生物医学工程系

牛金海 博士 副研究员

编译者

目录

、

2007-10-19

美国德州仪器（TI）－上海交大（SJTU BME）联合 DSP 实验室介绍

美国德州仪器（TI）－上海交通大学（SJTU） 联合 DSP 实验室成立于 2007 年 10 月，位于上海交大闵行校区，致力于 TI DSP 技术的推广，以及相关数字信号处理算法的研究与开发，为客户提供优质的产品与服务，涉及的技术领域有，无线通信，音频/视频信号处理，医学信号/图像处理，数字马达控制等。实验室研发与培训教师主要由上海交通大学青年教师承担，同时聘请了多位有企业工作背景的 DSP 技术专家为实验室的顾问。实验室的主要方向：

1) 开展 TI DSP 相关的科研工作
2) 培养/培训 TI DSP 相关的工程技术人员
3) 承接 TI DSP 相关的项目合作与开发

实验室主任老师介绍：

牛金海 博士 上海交通大学 副研究员；五年数字信号处理的相关开发经验，具有丰富的数字信号处理（DSP）开发经验，包括数字信号处理，采集，编程，优化相关外围设备的开发经验，动手能力强，具有丰富的项目开发经验与组织管理能力。

2006 － 现在 副研究员， 上海交通大学，生物医学工程系。

2003－2006 年 凯明信息股份有限公司 数字基带部（DBB）以及系统工程部（SE）， 从事 TD－SCDMA 移动终端 数字基带 ASIC 芯片的开发，职位为高级工程师。

2001－2003 年 华为公司上海研究所 数字信号处理部，从事 WCDMA 基站 数字基带 ASIC 芯片的开发， 职位为开发工程师。

1998 － 2001， 在上海交大昂立，华浦，韦博国际上海中心等职业培训中心。有十几个班，几百节课的培训经验（兼职）。

DSP 培训介绍

主讲老师介绍： 我们聘请交大的资深教师以及有 5 年以上企业 DSP 相关开发经验高级工程师主讲。主讲老师具有丰富的数字信号处理（DSP）开发经验，包括数字信号处理，采集，编程，优化相关外围设备的开发经验，动手能力强，

具有丰富的项目开发经验与组织管理能力。

颁发 TI 授权的培训证书

开课时间：常年招生，每月一期，循环授课，16 个学时；学不会可以免费参加下一期培训，结业颁发证书。培训资费：在校学生以及三人以上集体报名 8 折。

垂询电话：13651621236，jhniu@sjtu.edu.cn (牛老师)，

请访问 TI-SJTU 联合 DSP 实验室的网站了解最新信息：

http://life.sjtu.edu.cn/dsp_lab/index.html

课程介绍：

时间以及主题	内容简介	适合培训对象	培训费用
《TI DSP 基础》 16 学时	1、 TI DSP 特点，应用，分类等 2、 开发环境 CCS 使用等 3、 C6000DSP 的芯片架构等 4、 C 以及汇编编程	DSP 爱好者以及初学者	800 元人民币
《TI DSP 中级技术》 16 学时	1、 代码优化，线性汇编 2、 BIOS 实时操作系统 3、 TIMER,DMA 等外设以及扩展技术 4、 Boot loader 原理等	具有一定的 DSP 基础知识或者参加过 DSP 基础培训的学员	1200 元人民币
《TI DSP 在生物医学工程中的应用》 8 学时	1、DSP 在医学信号处理中的应用 2、DSP 在医学图像处理中的应用 3、应用实例介绍	在生物医学工程领域的研究技术人员，大学老师，高年级学生，研究人员	800 元人民币

第一章　绪 论

本节介绍了德州仪器的eXpressDSP技术倡议，同时也介绍了Code Composer Studio IDE的简单开发流程。

1.1 欢迎来到 eXpressDSP 世界

TI提供多种开发工具，可以快速实现基于DSP的应用设计，从概念到编码/编译，调试，通过分析，校正，并以测试。许多工具都是TI的实时eXpressDSP ™软件和开发工具策略的一部分，这对设计过程中快速起步及节省宝贵时间非常有帮助。TI的实时eXpressDSP软件及开发工具策略包括三个组件，使开发者能够充分利用的TMS320™系列DSP潜能：

Code Composer Studio IDE中强大的DSP集成开发工具

eXpressDSP 软件，包括：

— 可扩展性，实时软件基础：DSP/BIOS™的内核

— 应用程序互操作性和再使用标准：TMS320系列DSP算法标准

— 设计就绪代码对许多应用很常用，用它可以快速开始dsp的设计：eXpressDSP参考框架

越来越多来自第三方的基于TI DSP的产品，其中包括能被系统快速集成的eXpressDSP兼容的产品

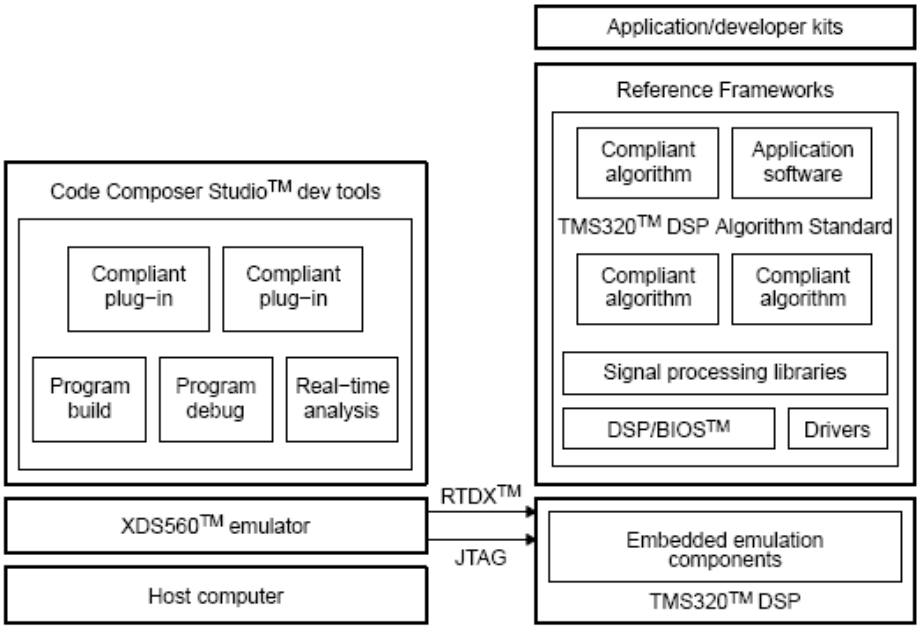

图 1-1　eXpressDSP软件和扩展工具

1.2 开发流程

大部分基于DSP的应用程序开发包括四个基本阶段：应用设计、代码创建、调试、分析与调整。本书将提供程序开发流程中基本的步骤和技术。

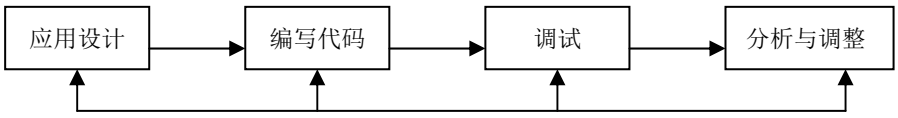

图 1-2 简单的CCS开发流程

第二章 快速入门

这一部分介绍了 CCS 的一些基本特征和功能，使你能创建和编译一些简单的工程。有经验的用户可以深入以下各节，获取所 CCS 各种特征的详细解释。

2.1 启动 CCS IDE

双击桌面上 CCS 的图标，就可以启动 CCS。CCS 启动时，系统默认配置一个软件仿真环境。如果需要配置特定的目标板，可以参考第三章的详细介绍。

CCS 所使用的重要的图标，手册全文将涉及到这些图标。

 Launches Code Composer Studio）

 重新构建工程（Rebuilds the project）

 增量构建（Builds the project incrementally）

 暂停执行（Halts execution）

 断点（Toggles breakpoint）

 运行（Runs project）

 进入（Single steps project）

 跳出（Step out）

 跳过（Step over）

2.2 创建新工程

按照以下步骤创建一个新工程：

1. 如果CCS安装在C:\CCStudio_v3.1，在C:\CCStudio_v3.1\myprogects文件下新建一个practice文件夹。
2. 把C:\CCStudio_v3.1\tutorial\target\consultant文件夹下的内容复制到新建的文件夹里。目标与当前CCS的配置有关。在使用CCS前必须进行配置，CCS里没有默认的配置。关于CCS配置的详细介绍请参照第三章。
3. 选择**Project->New**。
4. 在Project Name框里输入工程名：(例如:practice)。
5. 在Location框里输入或者浏览第一步创建的文件夹。
6. 工程类型默认为可执行的(.out)文件，目标设置为CCS当前的配置。
7. 点击 **Finish** ，CCS创建一个叫做practice.prj的工程文件。文件里存储了工程配置和工程所需要的各种相关文件。
8. 选择 **Project->Add files to Project**，把文件加到工程里。你也可以在左边的工程视

图窗口里右击工程，选择 Add files to Project。

9. 从你所创建的文件夹里添加 main.c, DoLoop.c, 和lnk.c(映射内存的连接命令文件)。浏览C:\CCStudio_v3.1\c6000\cgtools\lib\目录，为所配置的目标添加相应的rts.lib。
10. 你不必手工添加包含文件到工程里，因为编译程序的时候，程序会自动找到它们。在编译一个工程后，包含文件会出现在工程视图里。

2.3 构建程序

创建了一个功能程序后，你就可以构建（build）它。构建主要完成编译（compile）与连接（link）。第一次使用全构建（Build All）功能便可以构建工程，以后可以使用增量构建（Build the project incrementaly）。一个输出窗口将会显示构建过程和状态。当构建完成后，输出窗口将会显示Build complete 0 errors, 0 warnings。

当工程选项或文所有件发生改变后，执行Rebuild All命令重建工程。

2.4 加载程序

程序成功构建后，执行**File->Load Program**加载程序。加载过程是将上述构建成功，生成的可执行文件加载到目标板，目标板可以是软件仿真环境，也可以是硬件目标板。默认情况下，CCS集成开发环境将会在你的工程路径下创建一个Debug子目录，把生成的.out文件放在里面。点击**Open**加载程序。

注意： 如果你修改并且重新构建了工程，切记通过file－>Reload 重新加载程序。

2.5 基本调试

完成以下的练习，以便在实践中体会CCS的功能强大的调试器（debugger）。更详细的信息见第五章。

2.5.1 跳转到主函数（Go to Main）

选择**Debug->Go main**开始Main函数的执行。执行到Main函数出暂停，此时程序指针（黄色箭头）停在Main函数的左边空白处。这个区域被称为选择区域(select margin)。可以在这个区域设置断点（break point），探针(probe)等

2.5.2 使用断点 (Using Breakpoints)

把光标置于所需行上，按 F9 设置一个断点。此外，你还可以通过选择 Toggle Breakpoint 工具条按钮创建断点。设置断点后，一个红色图标将出现在选择空白区。再按 F9 或 Toggle Breakpoint 按钮将除去断点。

main.c 中，在 DoLoop 行设置断点(Input1, Input2, Weights, Output, LOOPCOUNT)
当程序暂停在 Main 函数处时，通过按 F5，选择 **Debug->Run**，或者选择 Run 工具条来运行程序。一旦程序运行到断点出，程序将挂起。

2.5.3 源代码调试（Source Stepping）

只有当程序执行挂起后才可单步执行。当程序在断点处挂起后，便可以单步执行程序。通过选择 **Source->Single Step** 按钮，单步执行到 DoLoop 函数中。多次单步执行观察执行结果。在 Single Step 按钮下，还有 Step Over 和 Step Out 可用。同样也可以使用汇编单步。因此，源程序单步逐行单步执行各代码，汇编单步单步执行各条汇编指令。汇编单步的详细信息见 5.2.1。

2.5.4 浏览变量（Viewing Variables）

在调试过程中，你应该查看变量的值以确保程序正常的执行。当 CPU 被挂起时，可以在 watch 窗口里观察变量的值。通过选择 **View->Watch Window** 可以打开 watch 窗口。Watch Locals 条目下显示当前执行的相关变量值。

当你连续单步执行整个循环时，变量的值会随之改变。此外，还可以通过把鼠标指针浮于变量上或把变量添加到 Watch1 Tab 里，来观察某个具体变量的值。有关变量和 watch 窗口的信息见 5.2.4 节。

2.5.5 输出窗口（Output Window）

输出窗口默认定位在屏幕下部，也可以通过 **View->Output Window** 来访问。默认情况下，printf 函数的输出显示在 Output 窗口，显示 Stdout 的内容和构建日志之类的信息。

2.5.6 符号浏览器（Symbol Browser）

Symbol Browser 功能使你能够通过单击访问工程里所有组件。选择 **View-> Symbol Browser** 可打开该窗口。Symbol Browser 有多重标签：Files、Functions、Globals

展开 Functions 标签树，显示工程里所包含的源文件。双击文件名或者函数标签将自动访问相应文件。Global 标签允许你访问工程里的全局符号。

有关 Symbol browser 的更多信息见 5.2.9 节。

到目前为止，你应该已经成功创建、构建、加载和调试了你的第一个 CCS 程序。

2.6 帮助文件介绍

CCS 通过 Help 菜单提供了许多帮助文件。选择 **Help->Contents** 按内容搜索，选择

Help->Tutorial 访问指南以帮助你了解 CCS 开发过程。

选择 **Help->Web Resources**，获取目前大部分帮助主题和其它的指导。PDF 帮助文件会提供具体特征和流程的信息。

通过 **Help->Update Advisor**，你可以获取更新资料和许多可选插件。

第三章 目标和主机设置

这一部分介绍了如何定义和设置单处理器或多处理器的目标计算机的特性,如何根据客户的需求定制集成开发环境备几个通用选项。

3. 1 设置目标主机

3.1.1 CCS 设置工具 (CCS Setup Utility)

这一部分介绍了如何定义和设置单处理器或多处理器的目标计算机的特性,如何根据客户的需求定制集成开发环境的几个通用选项

3.1.1.1 加入一个存在的配置

设置工具允许你配置软件，使得 CCS 软件能够在不同的硬件或软件仿真环境下工作。打开 CCS IDE 集成环境前，你必须选择一个合适的环境配置。

用户可以用软件提供的标准配置文件产生一个配置，也可以用自己的配置文件产生一个客户化的配置（参考在线帮助或者例程）。这个例子使用标准配置文件。

使用标准配置文件产生一个系统配置：

1. 双击桌面上的 Setup Code Composer Sutdio 图标，出现系统配置配置对话框。
2. 从 available factory board 中选择与系统匹配的标准设置。确定可用配置中是否存在与系统匹配的配置，如果不存在，你可以创建一个自定义的配置（参考在线帮助或者例程）。

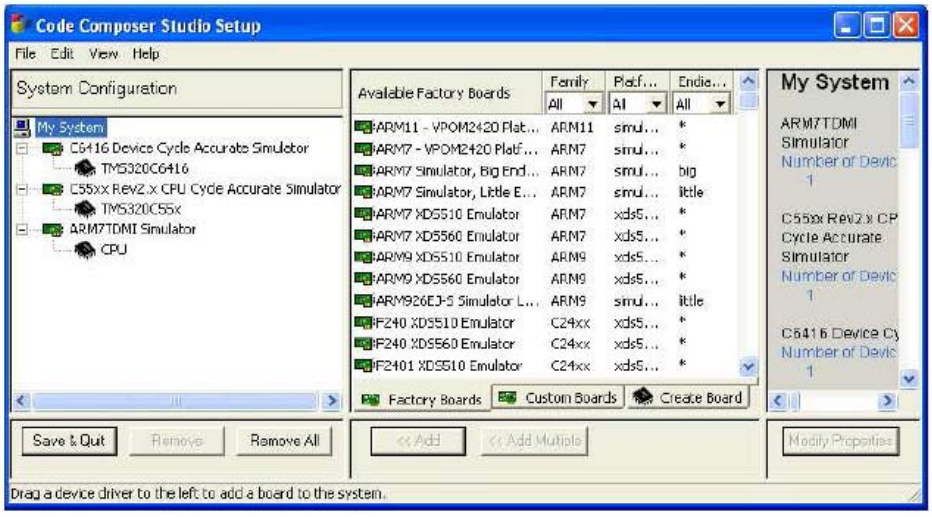

图 3-1 标准配置窗口

3. 单击选择的配置，然后单击 ADD 按钮将选择的配置添加到 system configuration 中。

这样选择的配置就出现在系统配置方框中 My System 图标下面。

如果你的配置包含多于一个的目标板，重复以上步骤直到你为每一个目标板选择了一个配置。

4. 单击 **Save & Quit** 按钮保存配置。
5. 单击 **Yes** 按钮，启动配置好的 CCS 集成开发环境。现在可以打开一个工程。关于打开一个工程，请参考这本书的第四章，或者是在线帮助和例程。

3.1.1.2 创建一个新的系统配置

要建立一个新的系统配置你可以从 Setup CCStudio 对话框开始。

你可以选择文件菜单中的 Remove All ，从而打开一个空白配置的工作环境。（你也可以打开一个与目标配置相近的标准配置的环境，在这种情况下，加载了启动配置后从以下第三个步骤开始改动配置）

1. 在系统配置框（System Configuration）中选择 My System 图标。
2. 在 Available Factory Boards 方框中，选择一个目标板或软件仿真器，代表你的系统，用鼠标拖动选择的目标板到 My System 下面，或者是按 Add 按钮。你可以通过 Family，Platform 和 Endianness 过滤属性，选择正确的目标板。你也可以拖动一个以上的目标板到 My System 下面。
3. 如果 Available factory board 中不存在你想要使用的目标板，你必须安装一个合适的设备驱动程序。（例如，第三方厂商提供的驱动程序或者你想使用 CCS 以前版本的驱动程序）进入安装/卸载设备驱动程序（选择 Help－>Contents—>Code Composer Studio Setup—>How To Start—>Installing/Uninstalling Device Drivers）继续完成你的系统配置。
4. 单击你刚刚加入的处理器类型，按照下列步骤打开 Connection Properties 对话框：

   － 右击 **system configuration** 方框中的处理器类型，从 **context menu** 中选择 **properties**。

   如果你选择了现在的处理器，选择的属性会出现在处理器属性对话框中。

   － 选择系统 **system configuration** 方框中处理器类型，然后选择最右边的 **Modify Properies** 按钮。
5. 编辑 connection properties 对话框中的信息，包括 connection name 和 data file，和 the connection properties。
6. 开始执行的 Gel 文件，处理器属性对话框中包含 Master/Slave 值、初始执行模式、BYPASS 名和比特数等信息。右击目标处理器，从列表中选择属性，这样就打开了处理器属性对话框。其它一些属性可能会有用，这取决于你的处理器。当配置软件仿真器时，许多属性已经有对应的默认值。

要获得更多的关于配置连接和处理器属性对话框，请参考在线帮助（Help－>Contents—>Code Composer Studio Setup—>custom setup）。

3.1.1.3 创建多处理器配置

最常见的配置包含一个软件仿真器或者是有单个 CPU 的单目标板。但是，你可以按照下列方法创建更加复杂的配置：

- 连接多个硬件仿真器到你的电脑，每个硬件仿真器都有自己的目标板。

- 连接多个目标板到一个单独的硬件仿真器，用特殊电路连接板上的扫描路径。
- 在单独的板上建立多个CPU，CPU可以是同一类型也可以是不同类型（例如,DSPs 和微处理器）。尽管一个CCS配置环境可以对应一系列的目标板，事实上，每个目标板要么是单个CPU仿真器，要么是单个硬件仿真扫描链，它们能连接到一个或者多个带多个处理器的目标板上。与目标板联系的设备驱动程序必须能够驱动扫描链上的所有CPU。

更多的信息可以参考在线帮助（Help－>Contents—>Code Composer Studio Setup—>how to start—>configuring ccstudio for heterogeneous debugging）

3.1.1.4 启动 GEL 文件

通用扩展语言 GEL 是一种解释语言，就像 C 语言一样，GEL 函数能够配置 CCS 集成环境。也可以初始化目标 CPU。很多 GEL 函数应用广泛，用户也可以自定义 GEL 函数。

设置 Processor properties 对话框中的 GEL 属性，可以把 GEL 文件与处理器相关联。选中处理器，打开 processor properties 对话框，在下拉菜单中选择属性。

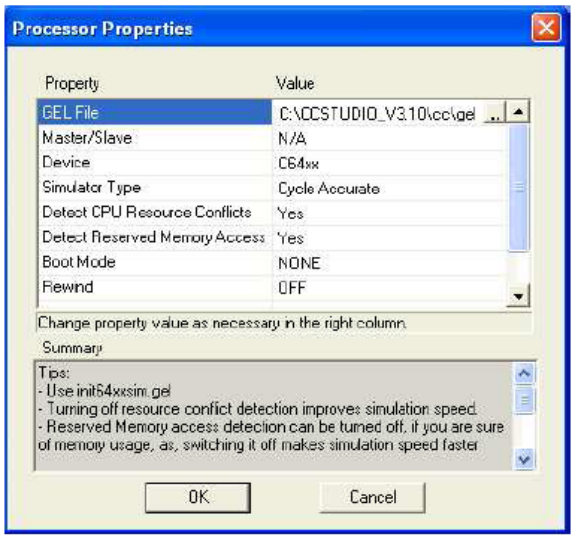

图 3-2 GEL 文件配置

当启动 CCS 的时候，CCS 会扫描启动 GEL 文件，自动加载 GEL 文件中的 GEL 函数。如果 GEL 文件中有 Startup 函数，这个函数就会被执行。例如，GEL Mapping 函数能够创建内存与调试器地址映射。

```
函数 Startup ( ) {/*Everything in this function will be executed on startup*/ gel_mapon();
gel_mapadd(0,0,0xf000,1,1);gel_mapadd(0,1,0xf000,1,1);}
```

GEL 文件是异步执行的，换句话说，GEL 中前一个指令执行完毕才执行后一条指令。更多信息请查看 CCS 在线帮助。选择 Help－>Contents—>Creating Code and Building Your Project－>Automating Task with General Extension Language）。

3.1.1.5 设备驱动器

设备驱动器是一个专门用来和主机通信的软件模块。每个驱动文件定义了一个特定的目标配置：一个目标板和硬件仿真器或者软件仿真器。设备驱动程序由德州仪器 TI 或者由第

三方提供。

Available factory boards pane 中列出的每个目标板或者是软件仿真都有对应的设备驱动程序。CCS 集成环境不支持自己创造设备驱动程序，但是 TI 公司或者其第三方会将设备驱动程序分发在预安装程序中。

3.1.2 并行调试管理器（PDM+）

在多处理器配置中，CCS 便启动一个专用的控制器即并行调试管理器（PDM+）。

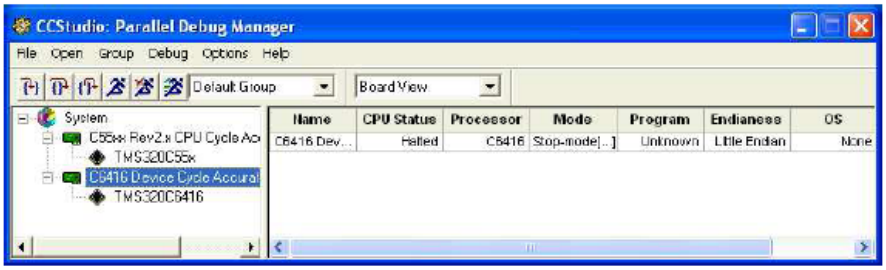

图 3-3 并行调试管理器

并行调试管理器允许分别为每一个目标设备打开一个 CCS 集成环境。PDM 控制能够实现设备的并行控制。

PDM 与以前的版本几点不同：

- 右击界面右边面板中的处理器，用户可以连接正在工作中的目标板或断开连接。
- 界面能够展开处理器的浏览图The interface allows an expanded view of processors，用户可以通过group，cpu，board分类过滤属性。
- 处理器图标显示红色（在左边的面板中）表示处理器未连接到系统或着它已经更新了状态信息。
- 你可以将处理器放入一个松散—联系（loosely-coupled）组中，（例如，处理器不全在同一个物理扫描链中）。选择工具栏中的第二个下拉列表组视图 （Group view）和PDM左边面板中的system显示哪些组是同步的。

注意：

当处理器工作在同一个物理扫描链（physical scan chain）上时全局断点才起作用。

如果您需要更多关于并行调试管理器的信息，请查看在线帮助Help－>Contents—>Debugging—>Parallel Debug Manager.

3.1.3 连接/断开连接（Connect/Disconnect）

CCS 集成环境使用功能 Connection/Disconnect,使连接目标，断开与主机的连接变得容

易。Connection/Disconnect 允许你断开与硬件目标的连接，甚至重新连接硬件目标时可以装载前一次的 Debug 状态。

在默认状态下，当控制窗口（Control window）打开时，CCS 集成环境不会试图连接目标。

单击菜单中的 **Debug－>connect**，就可以连接目标。在 **Options－>customize－>debug properties** 中可以更改连接的默认属性。

当目标板的状态发生改变，状态条会显示一个帮助图标。当目标板断开连接时，状态条会显示断开连接和目标板最后的执行状态（例如： halted,running,free running or condition）. 当目标板在连接状态，状态条会显示目标板 stepping（into，over，or out）和引起暂停（软件或硬件）的断点的类型。

当连接到目标板时（除了第一次连接），Debug 菜单中 restore debug state 的选项是可用的。选择这个选项可以使每个无效断点断开连接。你也可以按 F9 或者在右击下拉菜单中选择 Toggle Breakpoint 使它们复位。CTool jobs 和 Emu analysis 中的断点不被激活。

如果并行调试管理是开启状态，你可以右击 **Name** 列下的相应的目标设备，从而连接到目标设备。

如果您需要更多关于 Connection/Disconnect 的信息，请查看 CCS 在线帮助 Debugging->Connection/Disconnect。

3.2 主机 IDE 用户化定制（Host IDE Customization）

当 CCS 配置好并启动后，你可以根据客户需求配置几个通用的 IDE 选项。

3.2.1 默认的颜色与字体（Default Colors and Faults）

选择菜单 **Option－>Customize - >Font->Editor Font** 以及 **Option->Customize->Editor Color** 。允许客户修改编辑器（CodeWright text editor）的外观

选择菜单 **Option－>Customize - >Font->Tools Font** 以及 **Option->Customize->Tools Color** 。允许客户修改 IED 工具窗口的外观。

3.2.2 默认键盘快捷键（Default Keyboard Shortcuts）

默认的 ide 中有预先设定的键盘快捷键超过 80 个，这些快捷键可以被修改。也可以创建新的键盘快捷键，用来从文件窗口中调用编辑或者调试命令。分配键盘捷径过程如下：

1. 选择 **Option—>Customize。**

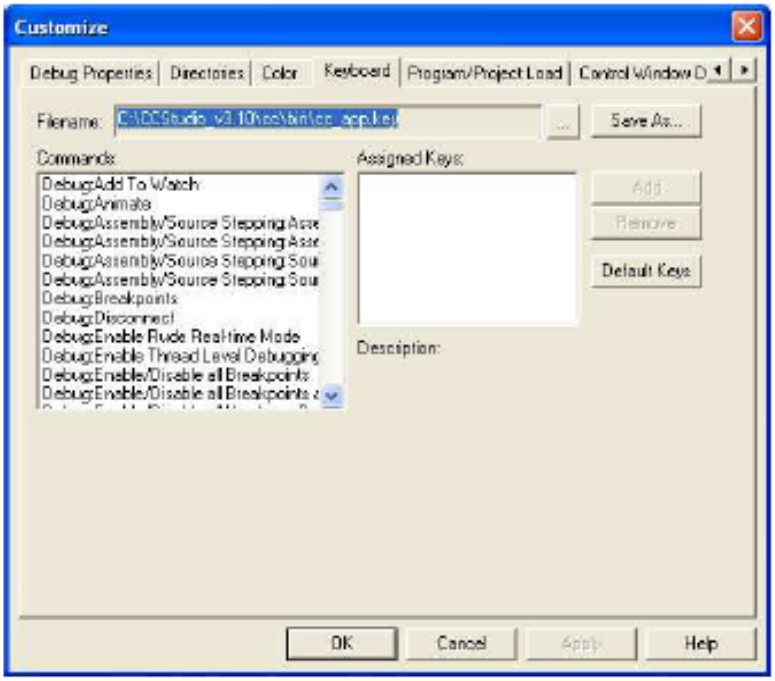

图 3-4 修改键盘快捷键

2. 在 Customize 对话框中，选择 Keyboard 标签查看下列选项：

－ **Filename 文件名** 在默认状态，显示标准的键盘快捷键文件。要加载先前的键盘配置文件（*.key）,键入路径和文件名，或者浏览文件。

－**Commands 命令** 选择你想分配给键盘快捷键的命令。

－**Assigned keys 已分配的快捷键** 显示已被分配了命令的键盘快捷键。

－**Add 添加** 点击 Add 按钮，给一个新的键序列赋值，调用选中的命令。在 Assign Shorcut 对话框中，键入新的键序列，然后单击 OK.

－**Remove 清除** 为了清除某一个命令的特殊的键序列，选择 Assigned keys list 中的键序列，然后单击 Remove 按钮。

－**Default keys 默认快捷键** 马上转到默认的键盘快捷键。

－**Save as 另存为** 单击 Save as 按钮保存你的键盘配置方案。在 Save as 对话框中，打开本地浏览，打开你想保存的位置，为配置文件命名，单击 Save。

3. 单击 **Ok**，退出对话框。

3.2.3 其它 IDE 用户化定制（Other IDE Customizations）

通过选择 **Option—>Customize—>File Access**。可以配置 File 菜单中最新打开过的 File 或 Project 的个数

选择 **Option—>Customize—>File Acess**，可以记录工程的当前路径。当你打开其它工程时，你能指定 IDE 打开当前工程的路径或是上一次打开的工程的路径。

选择 **Option—>Customize—>Control Windows Display**。可以设定标题栏中的信息的类型（处理器类型，工程名字，路径等等）

选择 **Option—>Customize—>Control Windows Display。**可以设定关闭选项。你可以指定当你关闭一个工程时，IDE 自动关闭所有窗口。或者当你关闭一个控制窗口时，关闭所有工程。

用 Codewright 定制代码窗口（详见章节 4.2.2）

第四章　代码创建

这章主要介绍建立代码的功能以及编译一个基本 CCS　IDE 工程的方法。

4.1 配置工程 （Configuring Projects）

一个工程包含了编译一个源程序或者库函数所需的所有信息，如：

- 源代码的文件名和目标库函数
- 代码生成工具选项
- 包含文件之间的从属关系

4.1.1 创建一个工程（Creating a Project）

下面的步骤教你如何创建一个或者多个工程（多个工程可以同时打开）。每个工程的名称必须不同。

一个工程的信息保存在一个单独的工程文件中（*.pjt）。

1. 打开 **Project->New**。显示出工程创建向导窗口。

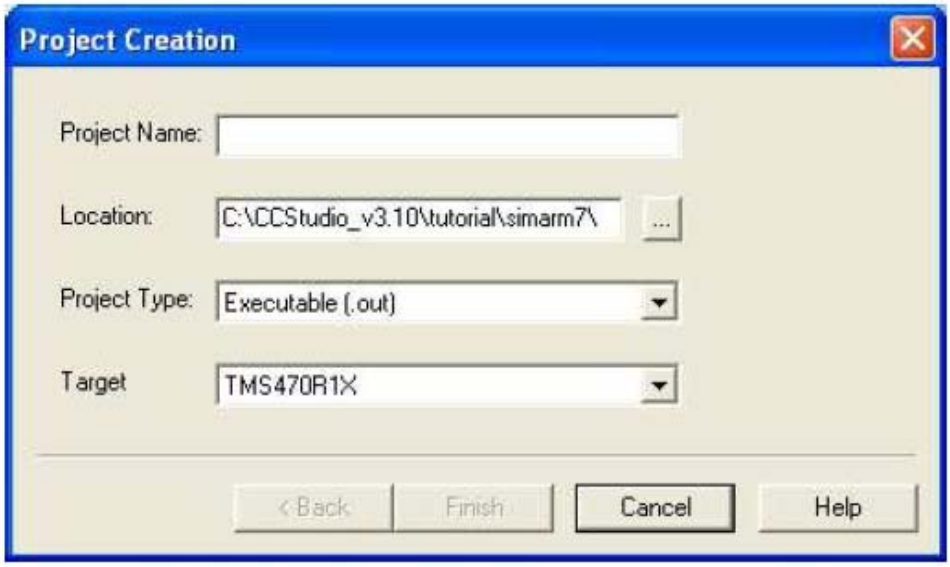

图 4-1 工程创建窗口

2. 在 **Project Name** 一栏中，输入工程名称。
3. 在 **Location** 一栏中，指定你希望工程文件保存的路径。编译器生成目标文件，汇编程序也存储在同一位置。你可以输入完整路径，也可以选择 Browse 指定存储路径。最好将不同的工程存储在不同的路径下。
4. 在 **Project Type** 一栏中，从下拉列表中选择工程文件的类型。可以选择执行文件（.out），也可以选择库文件（.lib）。可执行文件表示工程生成一个可以执行文件。

库文件表明你生成了一个目标库文件。

5. 在 **Target** 一栏中，为你的 CPU 选择目标板。当工具安装多个目标板上时，这个选择是必要的。
6. 点击“**Finish**”。一个工程文件就生成了。这个文件存储了你工程所需的所有文件以及工程设置。

这个新的工程和第一个工程配置（按首字母顺序）处于激活状态，继承了 TI 为 debug 和 release 配置提供的默认编译器和连接选项。

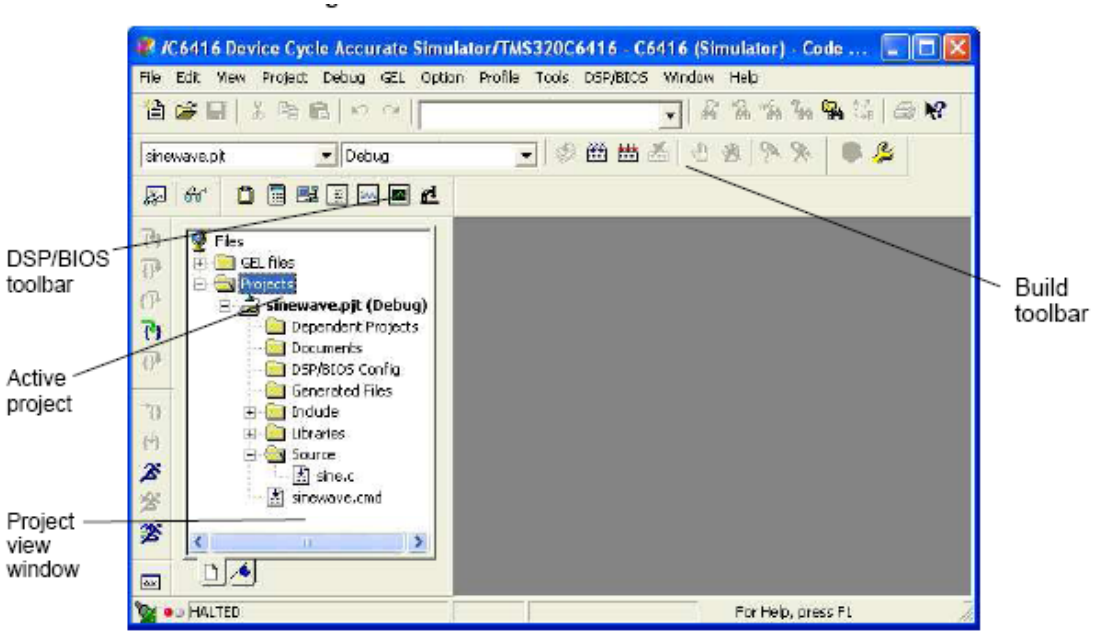

图 4-2　CCStodio IDE 界面

在创建一个新的工程之后，在工程列表中加入源文件，目标库文件和连接命令文件。

4.1.1.1 在工程中加入文件

你可以在你的工程中加入多个不同的文件和文件类型。如下图所示，在工程中加入文件：

1. 选择**Project→Add Files to Project**，或者工程视图（Project View）中的工程名上点击右键，选择加入文件到工程。显示加入文件到工程的对话框。

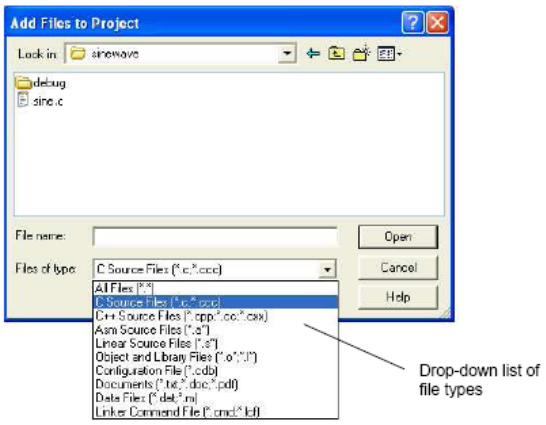

图 4-3 加入文件到工程对话框

2、在加入文件到工程对话框中，选择要加入的文件。如果文件不在当前目录中，改变文件路径。使用文件类型下拉选项，设置文件类型。

注意：

不要在工程中手动加入头文件或者库文件（*.h）。这些文件能够在编译过程中，扫描源文件的附件时自动加入。

---

3. 点击“**Open**”加入特定的文件在工程中。

当一个文件加入当前工程后，工程视图会自动更新。

工程管理器会将源文件，include文件，库文件和DSP/BIOS设置文件放入文件夹中。由DSP/BIOS生成的源文件放入Generated Files文件夹中。CCS IDE在编译程序时会按照以下路径顺序搜索文件：

- 包含源文件的文件夹
- 编译器和连接器选项内含的搜索路径中所列出的文件夹（从左到右）
- 可选 DSP_C_DIR(编译器) 和 DSP_A_DIR（汇编程序）环境变量（从左到右）定义中列出的文件夹

4.1.1.2 移除文件

如果你想从工程中删除一个文件，在工程视图（Project View）中右击文件名，选择从工程中移除。

4.1.2 工程配置 （Configurations）

工程配置定义了一系列工程层面的编译选项。在这个层面上设置的选项应用于工程中的每一个文件。

工程配置使你能够为每个不同的程序片断定义编译选项。例如，当 Debugging 你的代

码时，你可以定义 Debug 配置。当编译已完成的程序时，可以定义 Release 配置。

每个工程创建时都有两个默认设置：Debug 和 Release。还可以定义额外的配置。当一个工程刚创建或者一个工程刚打开时，工作区中的第一个配置（按首字母顺序）处于激活状态。

当你编译工程时，软件工具生成的输出文件置于配置类别的子目录下。例如，如果你在 MyProject 目录下创建一个工程，对于 Debug 配置的输出文件放在 MyProject\Debug 中。类似地，对于 Release 配置的输出文件放在 MyProject\Release 中。

4.1.2.1 改变激活的工程配置

单击选择工程工具栏中的活动工程配置（Select Active Configuration），在下拉菜单中选择一个配置。

图 4-4　配置 Toolbar

4.1.2.2 添加一个新的工程配置

1．选择**Project→Configurations**，或者在工程视图（Project View）窗口中，右击工程名称，选择配置。

2．在工程配置对话框中，单击加入。显示加入工程配置窗口。

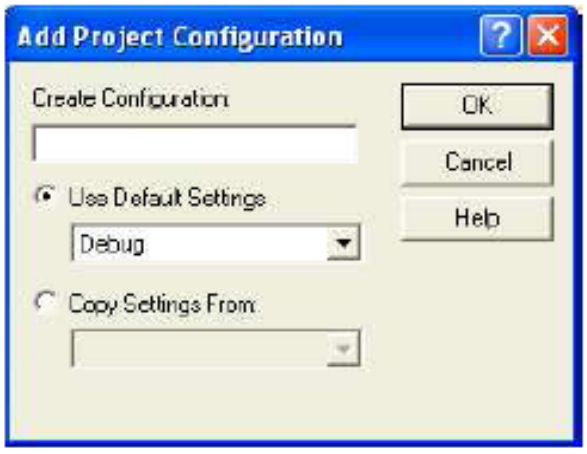

图 4-5 添加工程配置对话框

3．在Add Project Configuration对话框中，在创建配置一栏中指定新配置的名称，选择使用默认设置或者从已有的配置中拷贝设置，生成你的新配置。

4．单击 **OK** 保存你的选择，退出加入工程配置对话框。

5．单击 **Close** 退出工程配置对话框。

6．使用 Project 菜单中的编译选项对话框，更改你的新配置。

4.1.3 工程从属关系（Dependencies）

工程从属工具使你能够操作和编译更加复杂的工程。工程从属能够将一个大工程分割成多个小工程，然后使用这些工程从属创建最终的工程。子工程通常首先编译，因为主工程依靠这些子工程。

4.1.3.1 创建工程从属（子工程）

有三种方法创建一个工程从属关系或子工程。

(1)、从工程视窗中拖拽。将子工程放入目标工程中的目标工程图标上或者从属工程图标（dependent projects icon）上。你可以在同一个工程视图窗口中拖拽，或者也可以在两个同 时运行的 CCS 中的工程视图窗口之间进行拖拽。

(2)、从资源管理器中拖拽。

1) 打开 CCS 中的主工程。
2) 打开资源管理器。资源管理器和 CCS 必须同时打开。
3) 在 Windows 的资源管理器中，选择子工程的.pjt 文件。
4) 将这个.pjt 文件拖到 CCS 的工程窗口中，在移动的.pjt 文件之前将显示一个加号。
5) 将工程从属放入主工程文件夹中。

(3)、使用上下文菜单（the context menu）。在工程视图中，在一个下载后的工程中，右击工程从属图标。选择从上下文中，加入工程从属。在对话框中，浏览选择另一个工程的.pjt 文件。这个被选的.pjt 文件将成为一个下载工程的子工程。如果被选择的.pjt 文件还没有下载，将会自动下载。

4.1.3.2 工程从属关系设置

每个子工程都有分别的配置。另外，主工程针对每个子工程都有配置设置。所有这些设置都可以工程从属对话框中看到。打开这个对话框，可以从工程菜单中或者工程目录中选择工程从属。

4.1.3.3 修改工程配置

在Project Dependencies对话框中，可以修改子工程设置。前面提到，这个对话框可以通过**Project→Project Dependencies**进行访问。

如图 4-6 所示，你可以选择从你的配置中去掉某个子工程。在例子中，Sinewave.pjt 的配置文件，在编译中去掉了 zlib.pjt 文件。另外，还可以为这个配置选择特定的子工程配置。在 MyConfig 中，test.pjt 使用 Debug 配置编译而不是默认的 MyConfig 子工程配置。

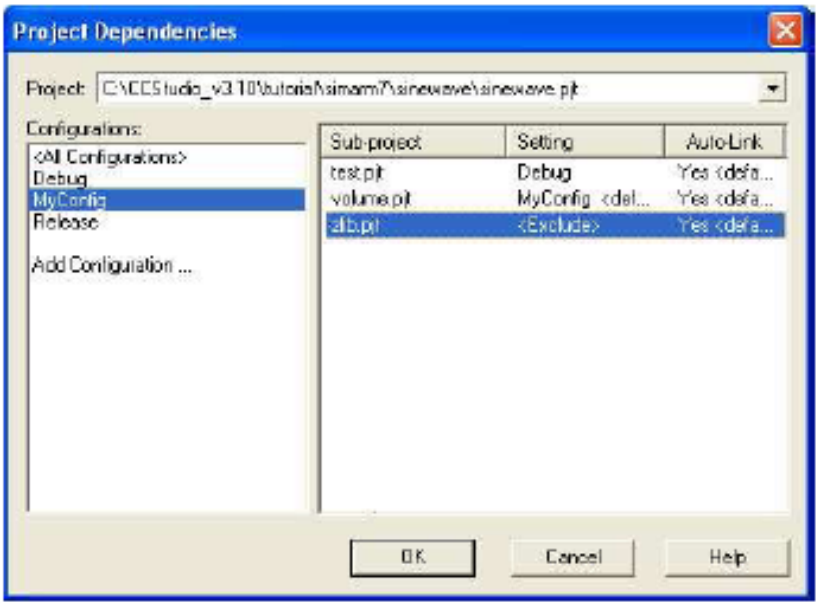

图 4-6 Project Dependencies 对话框

4.1.3.4 子工程配置

各个子工程分别有构建配置。对于每个主工程配置，你可以选择使用一个特定的配置编译每个子工程。使用工程（设置列下方）旁边的对话框，可以修改子工程配置。

4.1.4 制作文件（Makefiles）

CCS IDE支持使用外部makefiles (*.mak)，以及与之相关的外部编译设备和编译过程选择。与项目管理和用户程序创建有效的外部Make
使CCS IDE 能够使用编译文件编译一个程序，CCS工程必须包含一个编译文件，在CCS工程中加入一个编译文件之后，工程和其内容会显示在工程视图窗口中。选择**Project→Build和Project→Rebuild**所有指令可以用于编译所有程序。

1、在工程视图窗口中，双击编译文件，打开文件编辑。

2、修改编译文件指令和选项。

特定的对话框可以修改编译文件指令和选项。在编译文件工作时，正常的CCS 编译对话框不能使用。

可以创建多个配置，每个配置有自己的编译指令和选项。

注意：

限制和约束：在工程视图中可以加入和移除源文件。但是，在工程视图中的修改不能改变编译文件的内容，这些源文件不影响编译过程，也不在编译文件的内容中反映。同样地，编辑编译文件也不改变工程视图的内容。加入文件视图的，针对源文件的文件指定（file-specific）选项不起作用。Project®Compile File指令也不起作用。但是，当工程保存后，当前的工程视图状态也会保存。

注意：

在通过CCS IDE 指令使用编译文件编译程序之前，要设置必要的环境变量。运行宏文件

DosRun.bat可以设置环境变量。这个宏文件在C:\CCStudio_v3.1.如果你将CCS IDE 安装在其它路经之下，这个宏文件将在指定的路径下。

4.1.5 源控制集成（Source Control Integration）

工程管理器可以将你的工程连接到各种源文件控制提供者（source control providers）。CCS IDE 自动探测任何安装的可以识别的源控制提供者。

1、从 Project 菜单中，选择源控制（source control）。
2、从Source Control子菜单中单击选择提供者Select Provider。
3、选择你想使用的源提供者（Source Control Provider），单击确定。
如果在下拉菜单中没有源提供者，请确保你为提供者正确安装了客户端软件。
4、打开工程，选择 **Project→Source Control** 加入源控制。
5、在源控制中加入源文件。
6、可以在工程视图窗口中选择文件来加入或者移出源控制。在文件图标上右击，可以确定源文件连接到源控制上。

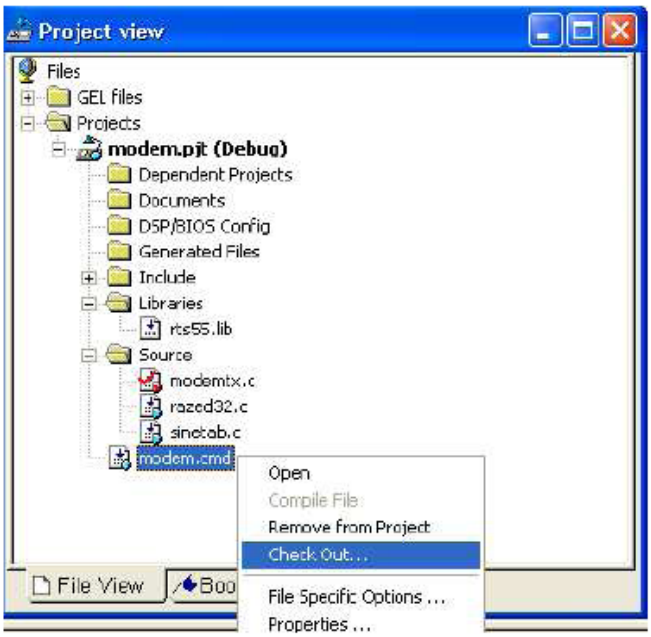

图 4-7 源控制集成

4.2 文本编辑器

4.2.1 查看和编辑代码 (Viewing and Editing Code)

在工程查看中双击文件名，在 IDE 窗口中会显示源代码

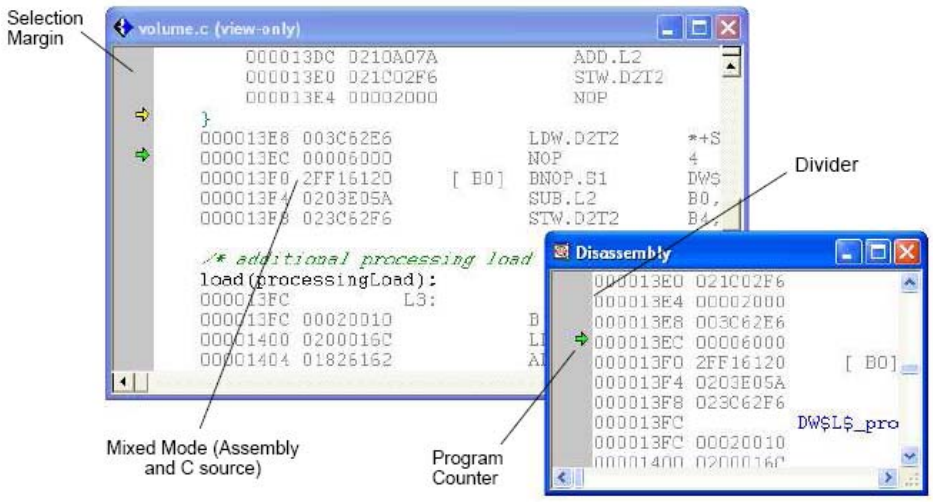

图 4-8 源代码窗口的基本单元

- **边白选择（Selection margin）** 默认情况下，边白选择标识显示在集成编辑器和反汇编窗口的左边。边白选择标识的彩色图标表示在这个位置设置了一个断点（红）或者探测点（蓝）。黄色箭头确定了程序计数器（PC）的位置。可以通过拖动分割线来调整页边空白选择的大小。
- **关键词（Keywords）** 集成编辑器的特点是突显关键词。关键词，注释，字符串，汇编指令，GEL 命令都被用不同的颜色突显出来。另外，还可以自己创建或者设置一套关键词并保存在关键词文档里（*.kwd）。
- **快捷键（Keyboard shortcuts）** 任何可以从文档窗口被调用的编辑命令或者调试命令，它们的默认快捷键都可被修改以及重新创建。在选项菜单里的自定义对话框中可以对快捷键的进行修改。
- **书签（Bookmarks）**可以在任一源文件的任一行设置书签来查询和维护关键位置。

4.2.2 定制代码窗口 (Customizing the Code Window)

IDE的文本编辑器（又叫CodeWright）允许用户自定义代码格式和特性。Option→Editor菜单中有语言，色度编码（ChromaCoding）体系和查看设置等附加选项。

- **语言** 用户可以将一套特性关联到一种文件类型（例如.cpp , .awk等等）。注意到，**Option→Editor→Language**下的文件类型列表与色度编码的列表是不同的。默认情况下，许多文件类型是关联到相应的词法分析的（例如，.h文件关联到C语法）。一些文件类型则没有对应的词法分析。
- **色度编码体系（ChromaCoding Lexers）** 一个色度编码体系存储了一些设置来改变程序设计语言词汇中不同成分的颜色。这些词汇包括标识符，大括号，预处理程序，关键词，算符，字符串以及注释。CodeWright 文本编辑器提供了以配置的大约 20 种特殊语言体系可以使用，包括几个适合 CCS IDE 的特殊词法体系（例如，GEL，CCS，C，DSP/BIOS 等等）。用户还可以在任意色度编码体系的对话框的右边点击新建或者保存按钮来创建新的编码体系。

- **查看设置** 定义了更多的适用于所有程序语言的一般特性，例如指定在所有语言中的所注释都是蓝色字体。但是，对于文本编辑器，词法分析定义了在注释前后所使用的分隔符。

CodeWright Menu Location	Configurable Settings and Options
Editor Properties (global settings): Option→Editor→Properties	Options for the editor, file loading, debug, selection margin resizing, tool tips, external editors, and backup (auto-save)
Settings for File Types (language properties): Option→Editor→Language	Language options and mapping, tabs and indenting, templates, coloring for code text, CodeSense, formatting for different file types, and comments
Lexer Settings (settings for language-specific lexers): Option→Editor→ChromaCoding Lexers	Identifiers, brace characters, color excluding for regex, adding new words (keywords, preprocessors, operators) and keyword defaults, language-specific comments, defaults for strings, number elements
View Setups (additional global settings): Option→Editor→View Setups	Showing line numbers and rulers, line highlighting, scrolling, line number widths, showing visibles (EOL, tabs, spaces, etc.), general color defaults, general font defaults
Advanced Text Processing Edit→Advanced, or right-click within text window and select Advanced	Caps to lower (and vice versa), inserting comments and functions, tabs to spaces (and vice versa), and other advanced editing options

表 4-1 代码文本编辑器

4.2.3 编辑器的文本处理功能的使用

文本编辑器包括了几种处理文本的附加功能。

- **区别和合并（Differencing and merging）** 用户可使用区别功能（File Difference between files）来比较两个相似文件的不同并显示出来。合并功能（(File Merge Files) 可以合并多文件。
- **标准表达式的维持（Support for regular expressions）** 选择 **Edit—>Find in Files or Edit—>Replace in Files**。除了一般的查找和替换功能，文本编辑器进行更为复杂的文本处理时，可使用正则表达式。例如，用户可在特定的目录中进行一个全局的替换。用户还可以使用已保存的查找结果，使用帮助下拉窗口（见下）可以使用户轻松建立标准表达式。

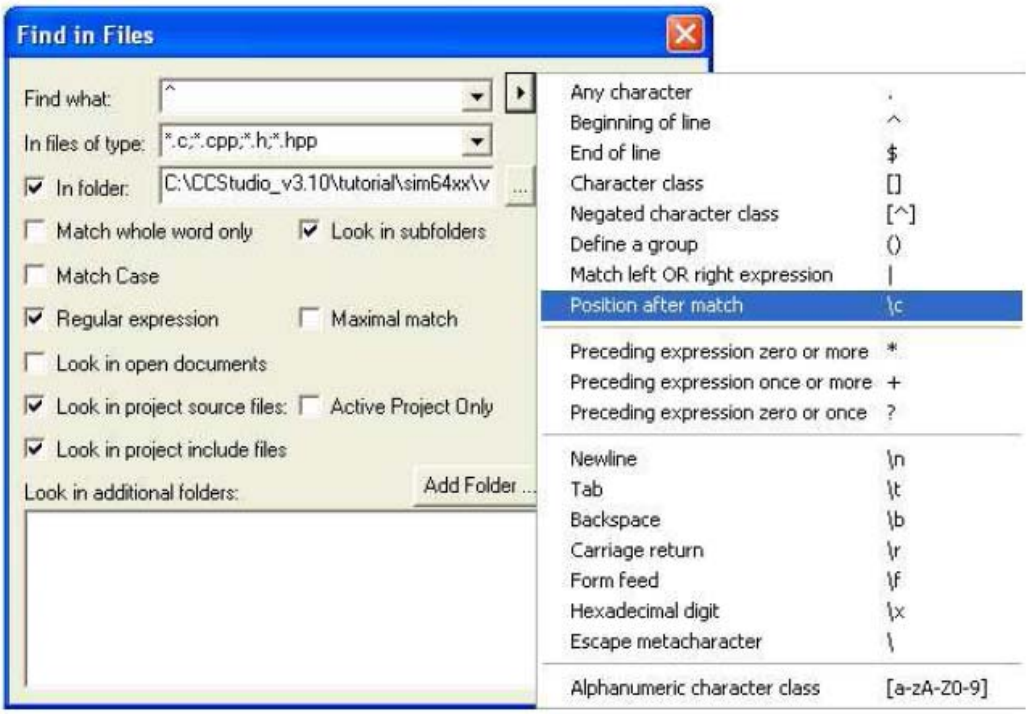

图 4-9 使用标准公式

- **选择性隐藏与显示代码** 选择性显示可根据选择参数确定某些种类的代码显示或隐藏。例如，用户可指定编辑器使用选择性显示功能来扩展和压缩某些种类的代码。或者可以通过选择适当的选项来隐藏所有的函数定义或预处理指令。当完成上述操作后，就会在页边空白出显示一个小图标，表示代码被隐藏了（见图 4-10）。点击图标可显示详细的代码。

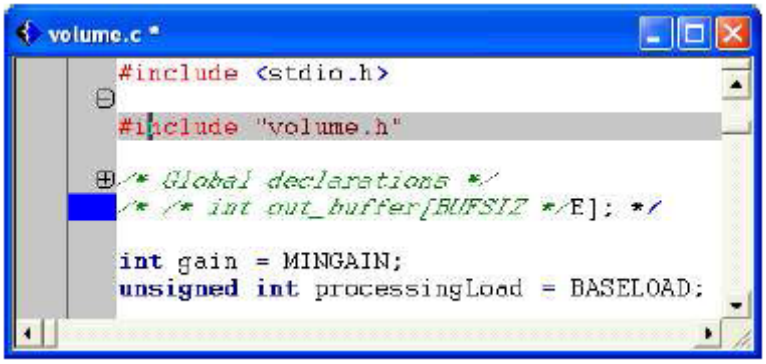

图 4-10 选择性显示

4.2.4 设定默认自动保存 (Setting Auto-Save Defaults)

文本编辑器能够定时的保存工作中的文件，以免由于系统崩溃造成的作业损失。选择**Option→Editor→Properties→Backup**，勾上复选框，则可使用自动保存功能。用户还可以选择保存的时间间隙以及指定备份文件的文件名和位置。除非另外指定了备份文件，在覆盖旧的备份文件时CCStudio会给出提示。

4.2.5 自动完成，工具提示和变量查看(CodeSense)

CodeWright 文本编辑器使用了一种自动完成引擎，称之为 CodeSense。当工具提示或自动完成功能激活时，在当前代码行的下面会出现一个图标，显示了 C，C++和 Java 代码的符号，函数参数和工具提示。工具提示也可用在变量查看中。

CodeSense 只适用与某些文件类型并且 CodeSense DLL 要被激活。

激活 CodeSense 的步骤：

1. 选择**Option→Editor→Language→CodeSense。**
2. 在左边的框中，选择正在工作的文件类型。
3. 确定CodeSense DLL在文件类型选择框的右边是激活的。（如果CodeSense不支持某种文件类型，复选框是不能选择的）。

在激活 CodeSense DLL 后，CodeSense 可用于：

- 列出与正在键入的符号有关的符号（函数，结构体，宏，成员等等）。
- 从上下文列表中插入符号到当前文件。
- 在列表中使用被选择符号的 Goto 按钮进入到该符号的定义中去。（Goto 按钮的相应快捷键是 Ctrl+G）
- 得到正在键入的函数的所需参数的工具提示列表。

- 当悬停在工具提示上时可自动显示符号的定义，或者当按下 Ctrl 或 Shift 时显示（取决与 CodeSense 的设置）。

CodeSense 的补词功能可帮助用户完成正在键入的符号。一旦用户输入了字符，可通过下述步骤来使用该功能：

1. 同时按下 Ctrl 与空格键调出对应上下文的符号的列表框来选择。这些符号都是以已经键入的字符开头的，右手边的柱状物是各个符号的定义。

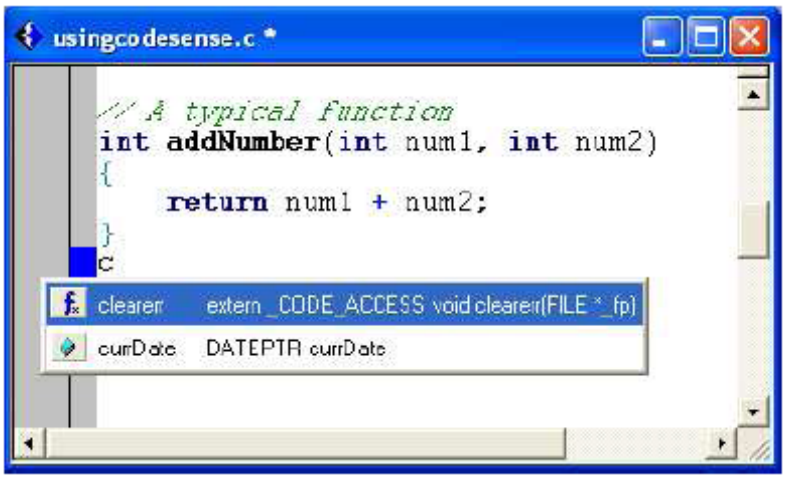

图 4-11 CodeSense

2. 在列表中选择适当的符号。按下已选符号对应的图标（Goto 按钮），在库文件源代码里显示符号的定义。用 Ctrl-G 也可以查看已选符号的定义。
3. 在下拉列表显示时按下回车键，所选符号就被完整的自动键入到文件中了。

4.2.6 使用外部编辑器 (Using an External Editor)

CCS IDE 支持使用外部文本编辑器来取代默认的集成编辑器。在配置好并激活一个外部编辑器后，只要新建一个空白文件或是打开一个已有文件就可使用。外部编辑器只能编辑文件，而调试程序还是要使用集成编辑器。可在 **Option→Editor→Properties dialog** 选择外部编辑器来进行配置。**常用的外部编辑器有，UltraEdit， SourceInsight 等。**

4.3 代码生成工具

4.3.1 代码开发流程 (Code Development Flow)

代码生成工具包括一个 C/C++优化编译器、一个汇编器、一个连接器以及其它各种工具。下图显示了如何在代码生成工具以及在其它工具的配合下生成代码。

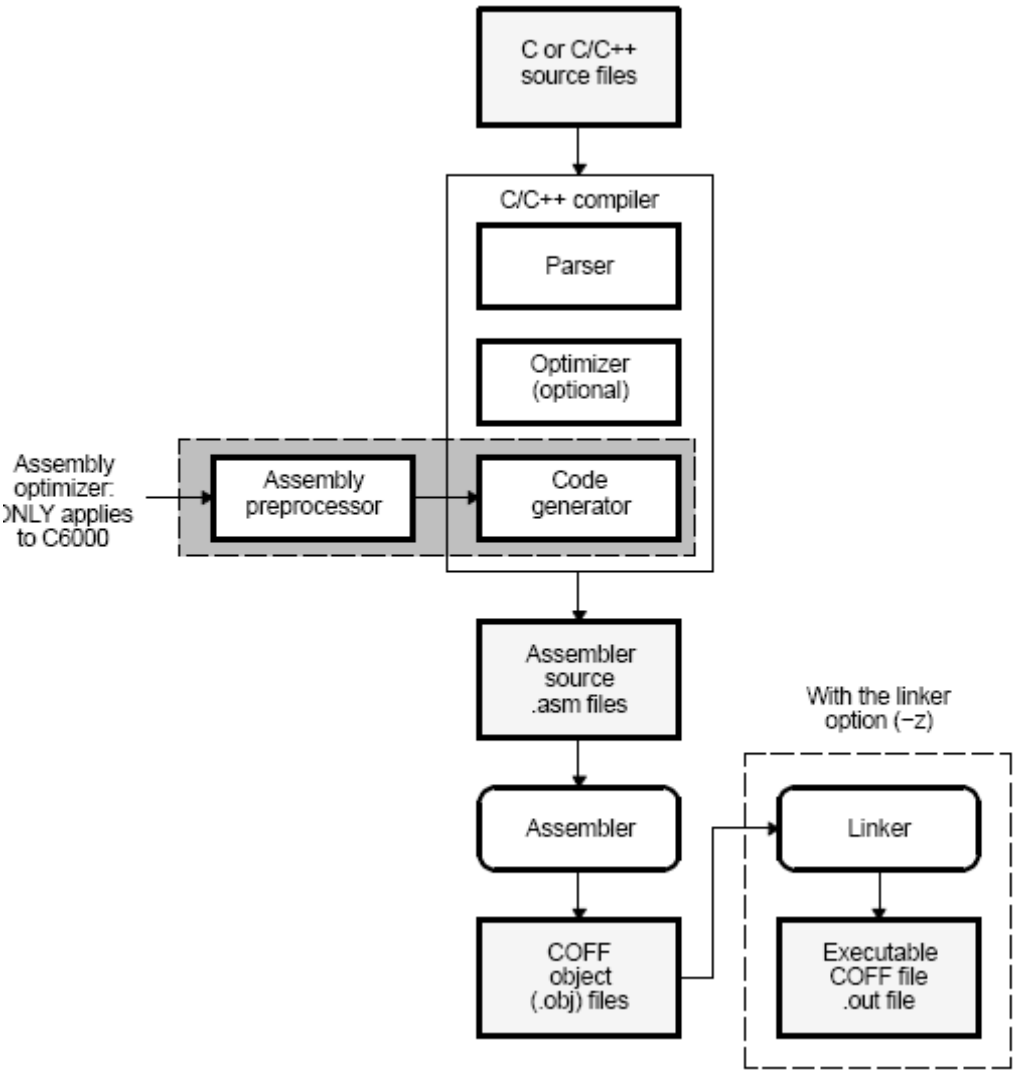

图 4-12 代码开发流程

4.3.2 工程创建选项 (Project Build Options)

工程创建选项是为使用代码生成工具提供的一个图形接口。一个 CCS 工程记录了一个目标程序或者目标库创建的所有必要信息。一个工程主要记录了以下几方面：

- 源代码和目标库的文件名
- 编译器、汇编器、连接器选项
- 包含了文件从属性

当你建立了一个工程，CCS 会调用适当的代码生成工具去编译、汇编、连接程序。

创建选项对话框详细列出了编译器、汇编器、连接器选项。这些选项的对话框列出了几乎所有的命令行选项。每一个选项并不意味着可以直接在顶部的可编辑文本框中直接写入，任何一个选项每一个目标配置有一个明确的设备设置选项。你可以通过阅读编译器指南或者汇编指南对你的目标寻求更多的信息。

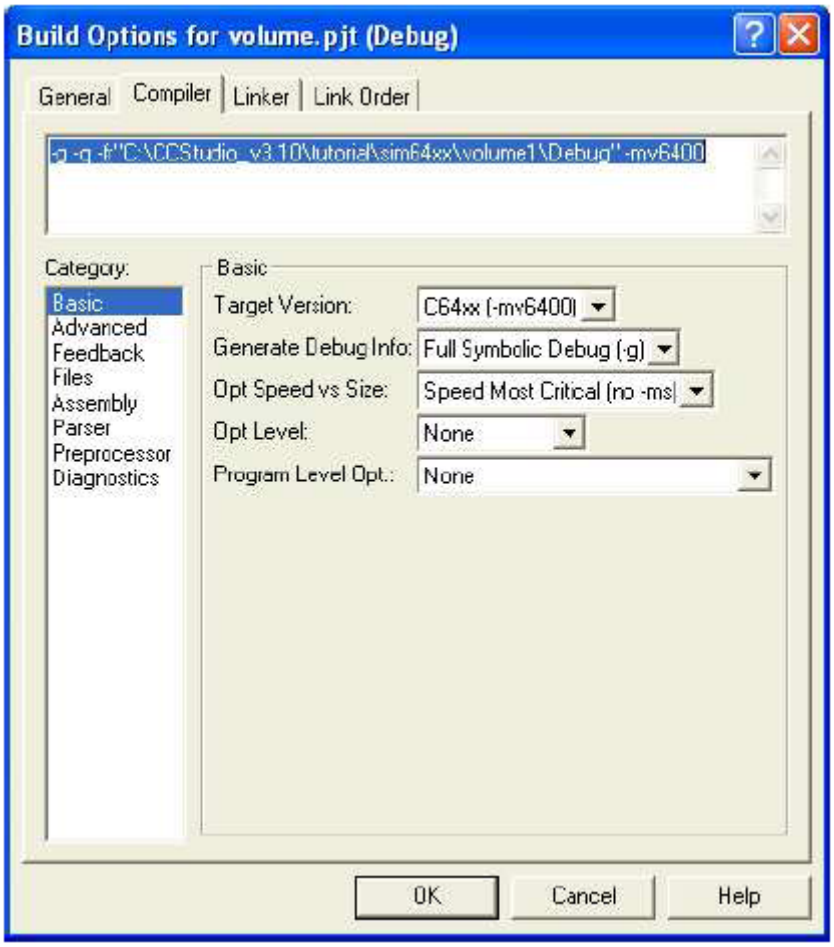

图 4-13 工程建立选项对话框

你可以在这选择在编译过程中要使用的编译器和连接器。

你的创建选项可以被设置为两个不同的层次，这取决于它需要的配置和使用频率。首先，你可以定义一个工程层次选项的设置，并把这种设置应用于工程中所有的文件。然后，你可以为各自的源代码文件定义文件细节层次的选项，以用来优化你的程序。

注意：因为很多选项一般都一起使用，所以你可以设置工程层次的配置，而不是反复的对各个选项进行同样的设置。你也可以通过在线帮助和指南寻求更多的信息。

4.3.2.1 设置工程层次的创建选项

- 选择 **Project→Build** 选项
- 在创建选项对话框中，选择适当的标签
- 选择当你编译你程序时要使用的选项
- 点击 **OK** 确认选择

4.3.2.2 设置文件-细节（File-Specific）选项

1．在该项目的视窗右击源文件名，然后从上下文菜单中选择具体文件选项
2．选择编译该文件时需要使用的选项，这将有别于工程级别的编译连接选项
3．点击 **ok** 来确认您的选择
4．所有变化只适用于选中的文件

4.3.3 编译器概述 (Compiler Overview)

C和C++编译器(用于C5000™ 和 C6000™)是全功能的优化编译器，能将标准的ANSI C程序转变为汇编语言。以下几部分描述编译器的主要特征。

4.3.3.1 与CCS集成开发环境的界面连接

以下特性允许用户与编译器进行接口连接

- 编译器外壳程序（**Compiler shell program**）。编译器工具包含一个外壳程序，它可以用来编译，汇编优化，汇编，而且连接程序只需单步。如需详细资讯，请参阅附合您配置的优化编译器用户指南的外壳程序部分。
- 灵活的汇编语言界面（**Flexible assembly language interface**）。编译器具有直接的调用规定，所以您可以直接输入汇编语言或C语言来相互调用。如需详细资讯，请参阅附合您配置的优化编译器用户指南的实时环境部分。

4.3.4 汇编语言开发工具 (Assembly Language Development Tools)

以下是汇编语言开发工具的列表

- **汇编程序** 汇编程序将汇编语言源文件转换成机器语言目标文件。机器语言是建立在通用目标文件格式(COFF)的基础上的。
- **记录（Archiver）** 记录允许您收集一组文件变成单个的记录文件，称作一个库。另外，记录允许您通过删除、替代、提取或增加成员来修改库。记录的其中一条最有用的应用是建立目标模块的库。
- **连接器(Linker)** 连接器将目标文件转变成单个的可执行的目标模块，当它创建了可执行模块，它就执行再定位并解决了外部引用。连接器允许（relocatable）COFF目标文件和目标库作为输入。
- **绝对列表器 (Absolute lister)** 绝对列表器允许连接的目标文件作为输入并建立.abs文件作为输出。您可以安装这些.abs文件来生成包含绝对地址而不是相对地址的列表。如果没有绝对列表器，生成这样一个列表需要很多手工操作。
- **相互参照列表器 (Cross-reference liser)** 相互参照列表器使用目标文件来创建一个相互参照清单，在连接的源文件中显示它们的标志、定义和索引。
- **十六进制转换实用程序 (Hex-conversion utility)** 十六进制转换实用程序将COFF目标文件转换成TI-Tagged、ASCII-hex、Intel、Motorola-S或者Tektronix目标格式。您可以下载转换文件到EPROM程序器里。
- **记忆-算术转换器实用工具 （Mnimonic-to-algebraic translator utility）** 仅限于TMS320C54x设备，这个工具用来转换汇编语言源文件。该实用程序接受包含助记指令的汇编语言源文件。它将助记指令转换成代数指令，生成一个包含代数指令的汇编语言源文件。

4.3.5 汇编器概述 (Assembler Overview)

汇编器将汇编语言源文件转换成机器语言目标文件。这些文件是通用目标文件格式(COFF)。

双通道汇编器做到如下几点：

- 处理文本文件中的源程序语句，生成可重定位的目标文件。
- 根据需要生成源文件列表，使用户可以控制该列表。
- 允许您将代码分成几段，并为每个目标代码段提供段程序计数器（SPC）。
- 定义并引用全局标识符，并根据需要附加交叉参照列表到源列表。
- 汇集条件块（Assembles conditional blocks）。
- 支持宏，允许您内联或在库内定义宏。

4.3.6 连接器概述 (Linker Overview)

连接器允许您通过有效地分配输出段到内存映射来配置系统内存，当连接器连接目标文件，它执行以下任务：

- 分配段到目标系统的配置内存。
- 重新定位符号和段，并为其分配最终的地址。
- 解决输入文件之间未定义的外部引用。

连接器指令语言控制内存配置，输出片断定义和地址连接。该语言支持表达式的指定和评估，您可以通过定义和创建一个内存模块来配置系统内存。指定的内存和段允许您做以下几点：

- 分配段到明确的内存区。
- 组合目标文件段。
- 在连接时间定义或再定义全局标识符。

4.3.6.1 基于文本连接器

文本连接器连接目标文件到一个单一的可执行的COFF目标模块。在连接器命令文件里的连接指令允许您结合目标文件片断，连接片断或符号到地址或内部内存范围，并且定义或再定义通用符号。如需详细资讯，请参阅代码生成工具的在线帮助。

4.3.7 C 或 C++开发工具

以下是C或C++开发工具的列表：

- **C或C++编译器** C或C++编译器能接受C或C++源代码和生成汇编语言源代码。一个外壳程序，一个优化程序和一个内部列表工具（interlist utility）是编译器的组成部分。
  - 外壳程序能使您一步完成编译、汇编和连接源模块。如果输入文件有.a以及 .sa为扩展名则安装程序会调用汇编优化程序
  - 优化程序优化代码，以提高C程序的效率
  - 内部列表工具能把交织C或C++源语句用汇编语言输出
- **汇编软件优化程序（只适用于C6000）** 汇编软件优化程序允许您输入线性汇编代

码而不需要考虑管道结构和寄存器分配。它接受没有分配寄存器或未列表的汇编代码。汇编软件优化程序分配寄存器并且使用循环优选法将线性汇编软件转换成高度平行的汇编软件，后者利用软件流水线操作。

- **库—构建工具（Library-build utility）** 你可以使用库—构建工具去构建你自己定制的运行实时支持库。标准的运行实时支持库函数是以rts.src rstcpp.src源代码形式提供的。运行实时支持函数的目标代码被编译成小端模式或者大端模式以及C编码或者C++编码到标准库里。运行实时支持库包含了ANSI标准的实时支持函数，编译工具函数，浮点运算函数以及被编译器支持的C I/O函数。
- **C++名字反转换工具（C++ name demangling utility）**. 通过对连接层次名字中的签名（Signature）进行编码，可以使C++编译器实现函数重载，运算符重载，安全类型的连接。C++编译器会将程序中的变量名、函数名转换成内部名称,这个过程被称作Name Mangling,反过程被称作Name Demangling。当你在编译文件或连接器输出中检查转换的命名时，想把一个转换过的命名和C++源代码的相应的命名联系在一起是非常困难的。C++ name demangling工具是一个将每个能检测到的转换过的命名转变成对应的c++源代码中的原始命名的调试工具。

4.4 创建 CCS 工程

4.4.1 从 CCS 集成开发环境开始

按下列步骤编译连接和运行程序

- 选择 **Project→Rebuild All** 或者点击 Rebuild All 工具栏按钮。在你工程中的所有文件将被重新编译，重新汇编以及重新连接。这些过程的信息将显示在底部的窗口中。在默认情况下，.Out 文件将在你当前工程的 debug 目录下生成。可以在选择配置工具栏中选择不同的目录改变路径。
- 选择 **File→Load Program** 再选择你刚才编译的程序点击打开。这个程序就被加载到目标 DSP 中并且打开了反汇编窗口在窗口中显示了一系列组成该程序的反汇编指令。
- 选择 **View→Mixed Source/ASM** 这允许你同步的查看你的 C 源程序和汇编代码的结果。在混合模式窗口中点击汇编指令。可以在 CCS 集成开发环境中按 F1 键搜索那些指令的帮助。
- 选择 **Debug→Go Main** 从主程序开始执行，在 MAIN 程序暂停。
- 选择 **Debug→Run** 开始执行程序。
- 选择 **Debug→Halt** 退出运行的程序。

4.4.2 外部制作

CCS 支持使用外部制作文件（Makefiles），支持相关联的 Make 工具对工程进行管理和使构建过程客户化。

为了使CCS集成开发环境能用制作文件编译连接程序，一个CCS工程创建时必须包括制作文件(makefile)。在一个CCS工程与制作文件关联后，这工程以及它的内容可以显示在

工程显示窗口中，并且 **Project→Build and Project→Rebuild All** 这些命令可以用来编译连接程序。

通过双击工程显示窗口中的制作文件的名字，打开该文件进行编辑。你可以通过一些特殊的对话框改进制作文件的编译连接指令和选项。在制作文件工作的情况下标准的编译连接选项对话框是失效的。多重配置是可以被创建的，即每个文件有自己的编译连接指令和选项。

注意：

限制和约束：源文件可以通过工程视图从工程中添加和删除。然而，工程视图中的改变并没有改变制作文件中的内容。这些源文件既不影响编译连接过程也不反映Makefile中的内容。类似的，编辑制作文件也不改变工程视图的内容。但加入工程视图的源文件的详细文件选项将失效。Project Compile File命令也同样失效。然而，当工程被保存后，当前工程视图的状态就被保护起来。

4.4.3 命令行（Command Line）

4.4.3.1 在命令行中使用 Timake 工具

Timake 工具在 CCStudio_v3.1\cc\bin 目录下，它可以在命令提示下在 ccs 环境外编译连接工程。这个工具还可以完成大批的工程建立。

调用 Timake 工具：

- 打开 DOS 命令提示符。
- 通过运行批处理文件 DosRun.bat 建立必要的环境变量。这个批处理文件必须在使用 Timake 工具前运行。如果你把 ccs 安装在 C:\CCStudio_v3.1 则这个批处理文件的路径为 C:\CCStudio_v3.1\DosRun.bat。
- 运行 Timake 工具。

要想了解 Timake 工具的更多信息请用在线帮助。

4.4.3.2 Makefiles

除了在 CCS 集成开发环境中选择使用外部 Makefiles，你也可以将一个标准的 CCS 工程输出到一个标准的 Makefile，这个 Makefile 可以通过使用各种标准的 Make 工具以命令行的形式建立。CCS 提供了一个标准的 Make 工具，这个工具在 DosRun.bat 文件运行后运行。

将一个标准的 CCS 工程输出到一个标准的 Makefile：

- 在 Project 工具栏的下拉菜单下的 Select Active Project 中选择想要激活的工程文件名以激活想输出的工程。
- 选择 **Project→Export to Makefile**
- 在 Exporting <filename>.pjt 对话框中，指定输出的配置，一些默认配置，你的 Make 工具的主控操作系统以及标准 Makefile 的文件名。
- 点击 **OK** 按钮接受你刚才的选择并且产生一个标准的 Makefile

要想了解更多的有关工程输出到 Makefile 的信息请用在线帮助。

4.5 可用的基础软件

4.5.1 DSP/BIOS

DSP/BIOS 采用了可扩展的实时内核，专为 TMS320C5000,TMS320C2000 以及 TMS320C6000 DSP 平台设计。DSP/BIOS 使用户可以更快的开发及发布复杂的程序，并减少开发及维护配置操作系统及控制循环的需要。因为多线程使实时应用程序非常清晰的分离了，一个使用了 DSP/BIOS 的程序更易于维护，新功能可以在不打乱实时响应的条件下附加进来。DSP/BIOS 通过 C2000,C5000,C6000 平台提供标准 API 接口来支持快速程序移植。

升级版的 DSP/BIOS API 及配置工具可以通过升级建议器来获得，在安装了升级版的 DSP/BIOS 后，用户可以通过组件管理器来指定用哪个版本的 DSP/BIOS 来给 CCStudio 调用。

4.5.2 芯片支持库（CSL）

片支持库（CSL）提供了 C 语言功能来配置及控制片上的外设。这可以简化在一个实时系统上运行算法的过程。目标是使外设更易于使用，减少开发时间，使程序可移植，硬件，以及在小范围内达到外设的标准化及实现兼容性。

4.5.2.1 CSL的优点

CSL 有以下优点

- 对外设编程提供标准协议。CSL 提供一个更高层次的程序接口给每个片上的外充。这包括数据类型和宏定义给外设的寄存器配置，以及运行每个外设的各种功能函数。
- 基本资源管理。基本资源管理是通过使用打开和关闭各种外设来实现的。这对支持多通道的外设特别有帮助。
- 对外设提供符号描述。作为创建 CSL 的一个附加优点，一个对所有外设寄存器的完全的符号描述以及寄存器范围被定义出来，也就是说用户可以使用描述在前两个 bullets 的更高层次的协议，这样可以更少的定制外设，使代码更容易移植到新版本的 DSP 上。

4.5.3 板支持库（BSL）

TMS320C6000 DSK 板支持库是一套 C 语言应用程序接口，可用于配置及控制所有板上的设备，使开发者可以在一个实时系统里获得算法函数。BSL 包括分离的模块，这些模块编译好并存放在一个库文件里，每个模块代表一个独立的 API 并且被一个 API 模块引用。每个设备都由一个 API 模块包括，这样决定了模块的间隔尺寸。除了 I/O 端口模块，它分为两个 API 模块：LED 和 DIP。

4.5.3.1 BSL的优点

设备更易使用，设备间获得一定程序上的兼容性，缩短开发时间，可移植性，标准化，

以及硬件抽象等。

4.5.4 DSP 库（DSPLIB）

DSP 库包括很多可由 C 语言调用，经过汇编优化，通用目的的信号处理，以及图像/视频处理子程序。这些程序专门用于计算性的增强型实时程序中，那里对运行的最佳速度要求非常严格。使用这些子程序，用户可以获得比用标准 ANSI C 语言编写的程序更快的速度。并且，采用提供的信号处理，图像/视频处理功能函数，DSPLIB 及 IMGLIB，可以明显缩短应用程序的开发时间。

可以通过以下参考资料获得更多关于 DSPLIB 的信息：

- TMS320C54x DSP Library Programmer’s Reference (SPRU518)
- TMS320C55x DSP Library Programmer’s Reference (SPRU422)
- TMS320C62x DSP Library Programmer’s Reference (SPRU402)
- TMS320C64x DSP Library Programmer’s Reference (SPRU565)

4.5.4.1 DSPLIB的优点

DSPLIB 包括了一般使用的程序。源代码也提供了，可供用户修改函数以搭配自己特定的需要。

包括以下特点：

- 优化过的汇编代码子程序
- C 和线性汇编源代码
- 可供 C 调用的子程序与 TI 优化的 C 编译器兼容
- 检测标准(周期和代码大小)
- 用参照的 C 模块来测试

4.5.4.2 DSPLIB 功能概览

DSPLIB 提供了一套可供 C 调用的高性能程序代码，在很广的范围内编写信号、图像、视频应用程序时可用于关键功能。

在 DSPLIB 中包含的程序功能分为以下编目：

- 适应滤波（Adaptive filtering）
- 相关性
- FFT
- 滤波及卷积
- 数学
- 矩阵运算
- 其它

4.5.5 图像及视频处理库（IMGLIB）

图像/视频处理库（IMGLIB）包括了很多可供 C 调用的，经过汇编优化的，通用功能的信号处理、图像、视频处理的程序。IMGLIB 只能在 C5500/C6000 平台设备上使用。这些程序专门为用于计算功能的增强型实时应用程序设计，对执行速度有很严格的要求。使用这些

程序，用户可以获得比用标准 ANSI C 语言编写的程序更快的速度。并且，采用提供的信号处理，图像/视频处理功能函数，DSPLIB 及 IMGLIB，可以明显缩短应用程序的开发时间。

可以通过以下参考资料获得更多关于 IMGLIB 的信息：

- TMS32C55x Imaging/Video Processing Library Programmer's Reference (SPRU037)
- TMS320C62x Image/Video Processing Library Programmer's Reference (SPRU400)
- TMS320C64x Image/Video Processing Library Programmer's Reference (SPRU023)

4.5.5.1 IMGLIB的优点

IMGLIB 包括了通用程序。提供源代码使用户可以修改以用于搭配自己的具体需要。特点包括：

- 优化过的汇编代码程序
- C 和线性汇编源代码
- 可供 C 调用的程序与 TI 优化的 C 编译器兼容
- 检测基准 (周期和代码大小)
- 参照 C 模式的逆向测试（Tested against reference C module）

4.5.5.2 IMGLIB功能概览

IMGLIB 提供了一套可供 C 调用的更高性能的程序，广泛用于编写信号、图像、视频处理程序里的关键功能。

包括在 IMGLIB 里的软件程序分为以下编目：

- 图像、视频压缩及解压缩
- 图像分析
- 图像过滤/格式转换

4.5.6 TMS320 DSP 算法标准组件

DSPs 是用 C 及汇编语言混合编程的，可直接访问硬件外设。由于性能上需要，DSPs 几乎没有标准的操作系统支持。不像通用目的的嵌入式芯片。但是，由于缺少标准，这就不可能使算法不经重处理而用于多于一个系统。DSP 算法重用是非常耗费劳动力的，所以开发一个基于 DSP 的市场产品时间也很长。

TMS320 DSP 算法标准（称为 XDAIS）定义了一套 DSP 算法的要求，使系统集成者可以快速的组合系统以用于一个或多个这样的算法。

4.5.6.1 XDAIS的范围

1、第一层包括编程指导方针，用于在所有 DSP 结构的各种算法，不必考虑程序的区域。几乎所有最近开发的软件模块都已遵循了这些指导方针，所以这层是标准化。

2、第二层包括规则及指导方针，使所有算法可以应用于单一系统。建立了标准使一个算法使用数据内存，以及用于外部识别器的名字，同时规则及算法被封装起来。

3、第三层包括专门用于 DSP 家庭的指导方针。对算法使用处理器资源没有现行的统一指导标准。这些指导方针描绘了使用各种结构的概要。或许实际应用上会对这些指导方

针发生偏离，但算法的出售者可以在相关文档或模块头里概要的说明这些偏离。

在上图（Figure 4-14)里的阴影盒子表示区域是符合这种规格的版本。

4、第四层包括各种垂直市场。由于各种商业本身固有的不同，似乎应该由市场的领导者来制定基于该垂直市场的接口标准及算法。如果每个独立的算法都有一个接口，标准将不能正确存在。在这层次上，任何算法遵守了由最高的三层定义的规则，则被称为适合 eXpressDSP 的。

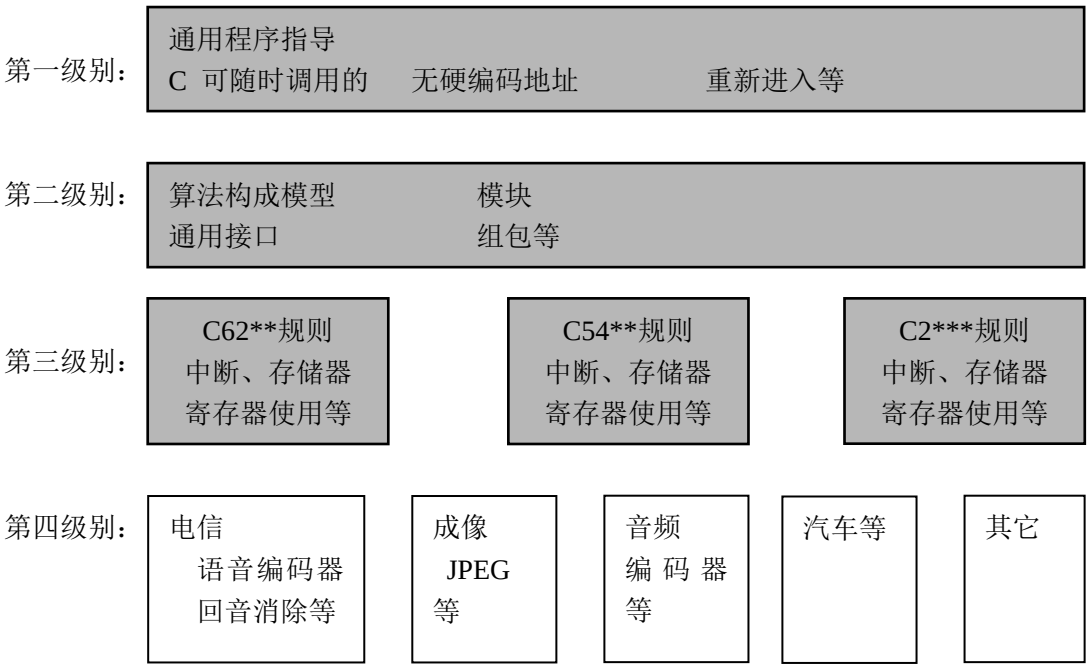

图 4-14 TMS320 DSP算法标准

4.5.6.2 规则及指导方针

TMS320 DSP 算法标准同时指定了规则及指导方针。规则必需遵守以使软件是适合 eXpressDSP 的。另一方面，强烈推荐遵循指导方针。

4.5.6.3 标准的需要

以下为 XDAIS 需要的元素：

- 从多个出售者得来的算法可以集成到单一系统。
- 算法是 framework-agnostic 的。也就是说，同样的算法可以实质上的就于任何应用程序或者框架下。
- 算法可以在完全静态或者完全动态的运行环境里开发。
- 算法可以以二进制形式发布

对算法的综合不需要结合客户的程序，但是，重新配置及重新连接则可能需要。

4.5.6.4 标准的目的

XDAIS 必需达到以下目的：

- 使开发者更易于遵守这些标准

- 使开发者验证这些标准的一致性
- 使系统集成者易于在 TI DSPs 间移植
- 使 host tools 简化系统集成者的工作，包括配置，性能表现模块化，标准一致，以及 debugging
- 在静态系统中几乎不产生 overhead

4.5.7 参考框架

eXpressDSP 软件的参考框架提供给使用了 DSP/BIOS 及 TMS320 DSP 算法标准的应用程序。用户首先选择最适合自己系统及将来需要的参考框架，接着使之适应这个框架，并广泛使用适用于 eXpressDSP 的算法。平常的元素如设备驱动器，内存管理及通道封装等已经预先配置好在框架里了，因此你可以集中精力于自己的系统设计上。参考框架包括已设计好的、可重用的 C 语言的源代码，基于 TMS320C5000 和 TMS320C6000 DSPs

参考框架软件及文档可以在 TI 网页上下载。它们不包括在 CCS 安装包里。

图 4-15 显示了标示了在目标 DSP 上参考框架的元素。

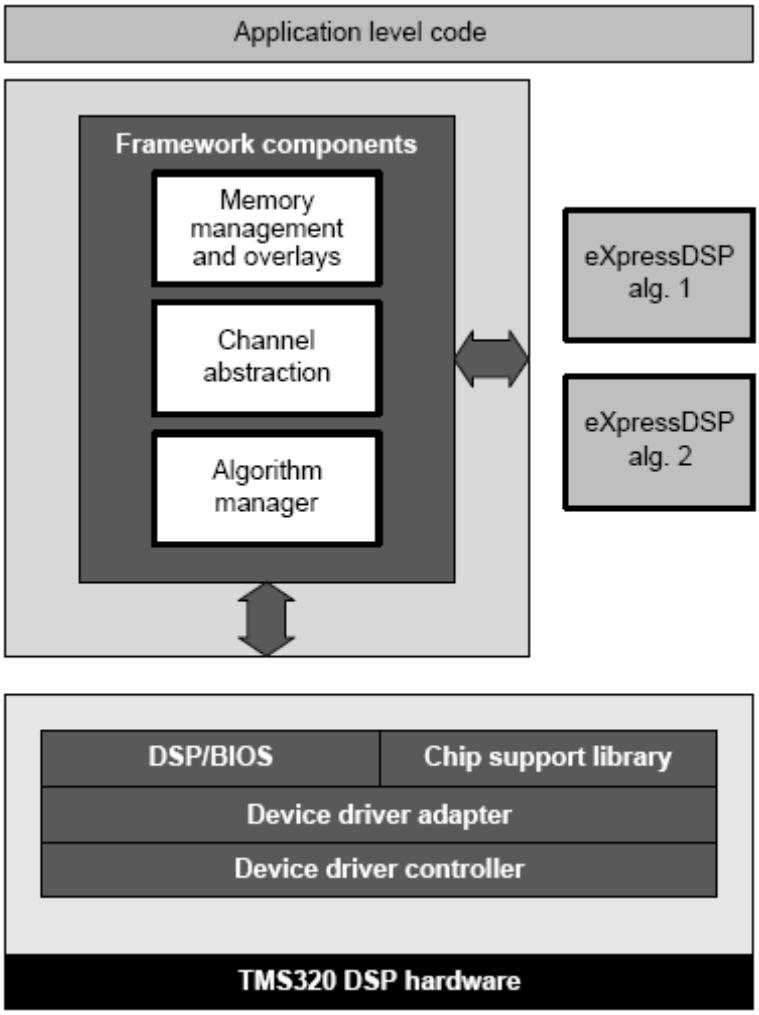

图 4-15 一个参考框架的组成元素

下面是框架中各组件的描述：

- **设备控制器和设备适配器** 引用框架中使用的设备驱动是基于标准驱动模型的。此模型提供设备适配器，并指定一个标准的设备控制器接口。如果使用特有的外设，设备控制器可能需要修改，但是设备适配器基本不需要修改。
- **芯片支持库** 设备控制器使用芯片支持库模块来支持外设。
- **DSP/BIOS** 这个扩展的软件内核是关于被引用到的框架是怎样根据它的需求均衡利用不同数量 eXpressDSP 底层构件的好例子。低廉的 RF1 框架使用相对较少的 DSP/BIOS 模块。对于未必很欣赏有如此多可选模块的设计者来说，除了带来明显的硬件资源节省，减少模块数量也使得设计方案的选择更清晰。
- **框架组件** 这些组件被设计用来管理所有系统资源。通道的抽象是其中的一个例子。每个应用框架都需要某种通道管理。然而，可以在预估的被使用通道数量基础上进行优化设计。对只含 1 至 3 个通道的简单系统来说，通道进程是由 DSP/BIOS HWI 和 IDL 模块来处理的。如果通道数量比较大的话，使用 SWI 模块将会更明智，虽然它会伴随更多板上硬件资源的耗用。对于通道动态改变的大型系统来说，TSK 模块显得更恰当。与通道管理器一样，算法管理器管理一些 eXpresDSP 体系下的算法。其它框架组件是用来处理内存覆盖机制的。在大多数内存有限的系统中，这是一项关键技术。从一个恰当的框架开始起步设计将简化许多开发选择。
- **eXpressDSP 体系算法** 在 TMS320 DSP 算法标准规则和准则（SPRU352）中给出了每种算法的具体规则和准则的细节规定。为了符合标准, 算法不能直接针对硬件外设，只能针对标准资源管理接口，即所谓的 IALG(内存管理), 和可选的 IDMA(DMA 资源管理)。在 TI 给出的范例中, 算法都比较简单，包括有限冲击响应滤波器(FIR)和音量控制器。用户可以用更符合 eXpressDSP 数字媒体软件标准的算法替代 TI 提供的范例，使得通用程序框架具体化为特定应用。
- **应用层代码** 最后一步是修改应用层代码。这一层代码利用特定应用领域的相关理论，使得产品细化得以实现。例如，单通道 MP3 播放器所需要的应用层代码和数字助听器所需要的是不同的。

4.6 自动化（项目管理）

4.6.1 使用通用扩展语言（GEL）

通用扩展语言是一种解释性语言，与 C 语言类似，它允许用户为 Code Composer Studio 创建函数。用户使用 GEL 语法来创建 GEL 函数，然后把它们导入 Code Composer Studio IDE. 可以利用一套 GEL 函数子集辅助工程建立，或者通过定制的 GEL 菜单自动打开和建立工程。下面是一个打开 volume 工程的 GEL 脚本范例：

```
/*
* Copyright 1998 by Texas Instruments Incorporated.
* All rights reserved. Property of Texas Instruments
Incorporated.
* Restricted rights to use, duplicate or disclose this
code are
* granted through contract.
```

```
*/
/*
* ======== PrjOpen.gel ========
* Simple gel file to demonstrate project management
capabilities of GEL
*/
menuitem "MyProjects"
hotmenu OpenVolume()
{
// Open Volume tutorial example
GEL_ProjectLoad("C:\\CCStudio_v3.1\\tutorial\\sim55xx\\volume1\\v
olume.pjt");
// Set currently active configuration to debug
GEL_ProjectSetActiveConfig("C:\\CCStudio_v3.1\\tutorial\\sim55xx\
\volume1\\volume.pjt",
"Debug");
// Build the project.
GEL_ProjectBuild();
}
```

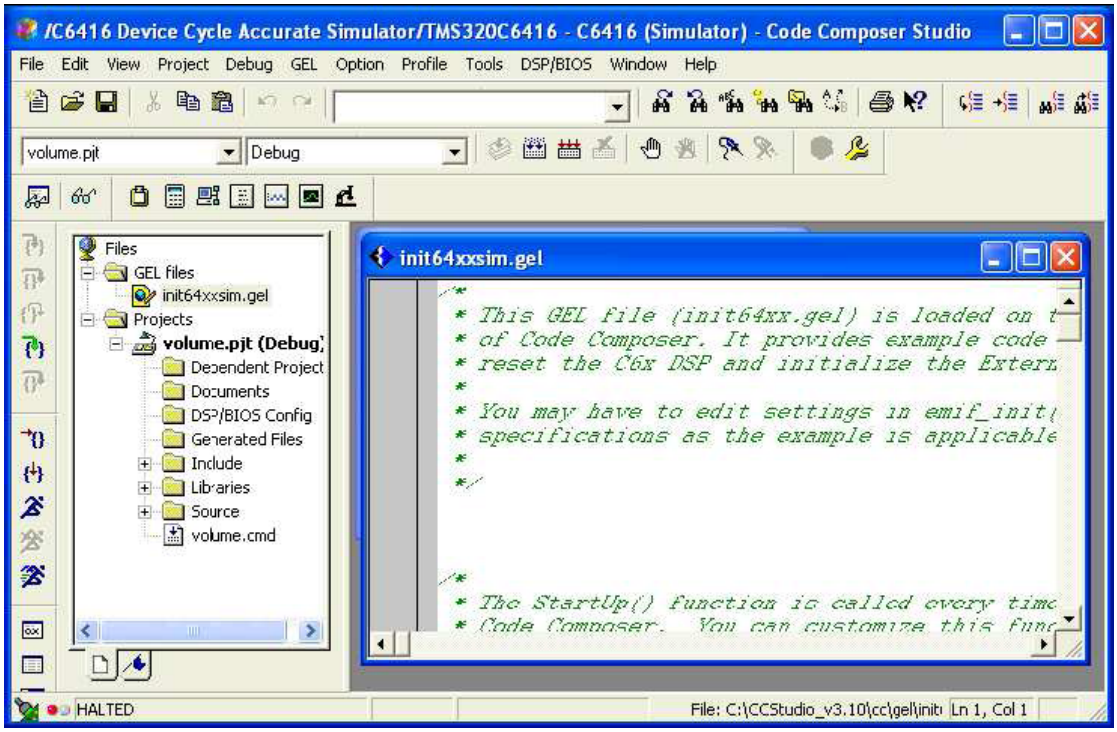

图 4-16 自定义 GEL 文件

4.6.2 脚本程序集 (Scripting Utility)

脚本程序集是一套集成 VB 或是 perl 脚本语言的 IDE 命令行。用户可以利用诸如 perl 或是 VB 脚本语言的所有特性并将它们与 Code Composer Studio 的 automation 任务相结合。

脚本程序集可以配置测试模式，打开和建立相应的工程，并装载执行。有许多这样可以用来建立和管理工程的脚本命令。脚本是同步的。

脚本通用程序是在 Update Advisor 中可供利用的附加特性（详见 7.2 节）

第五章　调试 (Debug)

这一节适用所有使用 Code Composer Studio IDE 的平台。然而，不是所有器件都能权限使用所有本节所讨论的工具。用户如果想知道可供使用的工具列表，可查看在线帮助和 Code Composer Studio IDE 提供的在线文档。

本节讨论 Code Composer Studio 包含的不同调试工具

5.1 建立调试环境

为了实现成功调试，调试环境需要按下面章节的介绍进行配置

5.1.1 设置用户调试选项（Setting Custom Debug Options）

几种调试选项是在 Code Composer Studio IDE 中可用户定制的。用户可以配置这些选项以便辅助调试过程或者符合自己的偏好。

5.1.1.1 调试属性标签

这个调试属性对话框在路径 **Option->Customize->Debug Properties** 下的用户化面板中找到。它允许用户在调试中禁用某些缺省的行为在线帮助。若要了解其它选项，请查看在线帮助。

调试属性中包含的行为如下：

- **自动打开反汇编窗口**：禁用这个选项将使得程序装载后反汇编窗口不再出现。缺省设置为打开。
- **自动跳至 Main 函数**：激活这个选项将使得在程序装载之后调试器自动跳至 Main 标号所在行。缺省设置为禁用。
- **当控制窗口打开时连接至目标器件**：控制窗口是整个 Code Composer Studio IDE 的交互接口。在运行 PDM 时可以打开多个控制窗口实例。用户遇到目标器件连接问题或不需要连接实际器件（例如些源代码时）时可以禁用此选项。缺省设置为禁用。
- **移除连接时剩余调试状态**：Code Composer Studio IDE 与目标器件断开时，依缺省设置它会移除断点。如果在调试过程中有错误，在重新连接器件时，Code Composer Studio 将会尝试再次移除所有断点。但是这种尝试将可能损坏某些目标器件。所以，TI 建议禁用这个选项，以避免重连器件时再次移除断点。
- **显示速度（Animation speed）**：显示速度是断点间的最小时间（秒为单位）。若从上次断点开始执行超过了此最小时间，程序执行将会重启。参见 5.2.1.1 节的更多细节。

5.1.1.2 目录

为了打开源文件，调试器需要知道应用调试时源文件位置。 调试器包含有工程中所有文件，当前目录下的文件的源文件路径信息，以及精确指向 Code Composer Studio IDE 的路径。

指向 Code Composer Studio IDE 的路径缺省条件下为空。所以，如果被调试的项目使用了打开的工程或是当前目录下没有的源文件，那么必须为这些文件单独设定路径。如果没有，由于调试器在需要引用此文件的地方无法自动打开文件，调试将会停止。此时，系统将会要求设计者手动寻找该文件。例如，在建立工程时包含库文件就是很典型的使用已打开工程中没有的源文件的例子。

目录对话框使得用户可以指定额外的搜索路径，这样调试器就可以依据该路径寻找被包含的源文件。

指定搜索路径的方法为，选中 **Option->Customize menu dialog** 中的 Directories tab。用户可能需要使用对话框中的滚动条来找到这个标签。

选项包括下列这些：

- **目录（Directories）** 目录列表显示被定义好的搜索路径。调试器依从上到下的顺序来搜索列出的目录。如果两个文件有同样的名字并且保存于不同的目录下面，那么出现在目录列表最高处的文件取得优先权。
- **新建(New)** 如果需要添加一个新的目录到目录列表，点击新建（new）。键入完整路径或者浏览合适的目录。缺省条件下，新目录将会被添加到列表的最底端。
- **删除(Delete)** 选中目录列表中的一个目录，然后点击删除（delete）将该目录移除出列表。
- **上移/下移(Move Up/Move Down)** 选中目录列表中的一个目录，然后点击上移（Move Up）将该目录上移或者（Move Down）将该目录下移。
- **查看子文件夹(Look in subfolders)** 用户可以激活调试器搜索列出路径中的子文件夹。
- **缺省 File I/O 目录(Default File I/O Directory)** 除了设置源文件目录外，用户还可以通过激活缺省的 File I/O 目录选项为 File I/O 文件设置缺省的路径。使用浏览按钮来寻找你希望作为缺省目录的路径。

5.1.1.3 程序装载选项

用户可以为通过选择 **Option->Customize menu dialog** 下的 Program/Project Load tab 为程序装载进行缺省设置。

它包括下面的缺省行为：

- **在程序装载时进行验证** 该可选框缺省设置为有效。这意味着 CCS 将确认（通过读取选定的内存单元）程序是否被正确装载。
- **在建立（工程）后装载程序** 当该选项被选中，在建立工程后马上就装载生成的可执行文件。这样目标器件就包含了工程建立后的当前符号信息。
- **装载时不设置 CIO 断点** 缺省情况下，如果用户的程序使用到 TI 实时库（rts*.lib），程序装载时，一个 C I/O 断点（C$$IO$$）将被设置。该选项使用户能够选择是否设置 C I/O 断点。C/O 断点是 C I/O 库函数如 printf 和 scanf 的正常调用所必须的。如果程序不需要执行 CIO 函数，那么 C I/O 断点是不必要的。当 C I/O 代码被装载到 RAM, Code Composer Studio 将设置一个软件中断。然而，当 C I/Od 代码被载入 ROM, Code Composer Studio 使用硬件中断。因为大多数处理器只支持较少数量的硬件中断，使用哪怕一个都对调试造成极大影响。可以通过在代码段中嵌入一个断点，并将其标签从 C$$IO$$重命名为 C$$IOE$$的方式，在 C I/O 代码装载到 ROM 时避免使用硬件中断。

- **装载时不设置程序结束断点** 缺省情况下，如果程序被连接到 TI 实时库（rts*.lib），程序装时将会被自动设置一个程序结束断点（C$$EXIT）。该选项允许用户选择是否设置程序结束断点。程序结束断点中止处理器的工作，使得程序不再往下执行。如果你的程序需要执行无限次数循环时，程序结束断点将是不必要的。如果程序结束断点代码载入 RAM, CCS 将设置一个软件中断。然而，当代码被载入 ROM 时，CCS 将设置一个硬件中断。因为大多数处理器只支持较小数量的硬件中断，使用哪怕一个都会对调试造成极大影响。可以通过在代码段中嵌入一个断点，并将其标签从 C$$IO$$重命名为 C$$IOE$$的方式，在程序结束代码装载到 ROM 时避免使用硬件中断。
- **装载新程序时禁用所有断点** 激活该选项将会在装载新程序前移除所有存在的断点。
- **装载工程时打开关联的工程** 缺省时，如果程序有主工程关联的子工程，所有子工程将随主工程一同打开。如果禁用该选项，主工程打开时子工程不会被打开。
- **装载工程时不扫描文件关联性** 为了能在工程增量建立方式下确定哪些文件必须被编译，工程必须对于每个源文件保存一个文件关联列表。每当建立一个工程时，关联树就会被创建。为了创建关联树，所有工程列表中的源文件将被循环的扫描用以寻找 #include, include, 以及.copy directives, 并且每个包含的文件名称将被添加到工程列表中。缺省状态下，当工程是打开的，所有文件中的工程都会被扫描相关性。如果该选项禁用，它就不会在代开工程时自动扫描相关性，而且打开工程的速度也将变快。

5.1.14 反汇编类型

在反汇编窗口中有可改变信息方式的选择设置。反汇编类型选择对话框允许你为调试部分选择特定的调试类型。

反汇编类型选择设置步骤：

- **Option->Disassembly Style**，或者在反汇编窗口中右击并选择 **Properties->Disassembly Options**。
- 在反汇编类型选择对话框中输入你的选择。

图 5-1 反汇编类型

- 点击 **OK**，反汇编窗口内的内容会按照设置立刻更新。

5.1.2 仿真（Simulation）

为了使仿真器按照实际硬件目标的方式运行，你可以设置内存映射（5.1.3 部分），管脚连接（5.1.4 部分），或者端口连接（5.1.5 部分）。

5.1.3 内存映射（Memory Mapping）

内存映射关系可以告诉调试器它能访问内存的哪个区域。内存映射关系可以随具体应用而改变。

当一个内存映射关系定义并且内存映射建立，调试器按照内存映射关系挨个检查内存入口。调试器不能进入被设置为保护状态的内存区域。

调试器按照软件内存映射关系检查内存入口而非按照硬件。调试器并不能阻止你的程序访问不存在的存储区域。

5.1.3.1 仿真器内存映射（Memory Mapping with Simulation）

仿真器利用预先定义的内存映射范围来作为被仿真的 DSP 硬件目标的内存设置。内存映射关系在一定程度上可以改变但是并不推荐这样做，因为仿真器的运行可能被大量有效的内存范围的改变而受影响。

5.1.3.2 利用调试器的内存映射（Memory Mapping Using the Debugger）

尽管在调试时可以改变内存映射关系，但是这是不方便的，因为使用者一般在调试前设定好内存映射关系，然后应用于其它所有的调试工作中。

添加一个新的内存映射范围步骤：

1. 选择 **Option->Memory Map**。

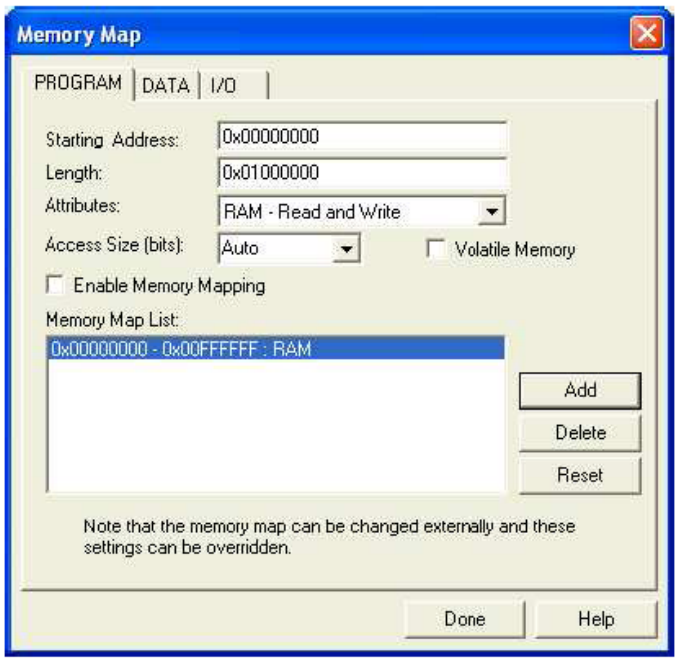

图 5-2 内存映射

2. 如果你的实际或仿真目标内存设置支持多类型，对于每一个类型内存映射关系设置对话框包含一个单独的标签（例如 Program,Data,and I/O）。选择你想改变的内存类型的标签。对于只含有一种类型内存的处理器，标签选择栏不出现。内存映射对话框提供以下选择：

- **激活内存映射（Enable Memory Mapping）** 保证内存映射关系激活箱核查无误，否则调试器认为你运行目标上的所有可设地址的内存（RAM）都是有效的。
- **起始地址（Starting Address）** 在起始地址输入框中输入新内存范围的起始地址。
- **长度（Length）** 在长度输入框中输入新内存范围的长度。
- **属性（Attributes）**在属性框中选择新内存范围的特性：读/写。
- **入口大小（Access Size(bits)）** 指定你的目标处理器的入口大小。你可以从下拉菜单中选择一个入口大小，也可以在入口大小框中键入一个数值。对于只支持一个入口大小的处理器没必要指定入口大小。
- **可变存储器（Volatile Memory）** 一般的，一个写通道包括读，修改和写操作。当可变内存设定在某个存储区域内，只通过一个写操作就可以访问那个区域。
- **内存映射清单（Memory Map List）** 显示内存映射关系的清单。
- **添加（Add）** 添加一个新的内存范围到内存映射关系清单中。
- **删除（Delete）** 在内存映射关系清单中，选择要删除的内存映射范围并单击 Delete 按钮。你还可以通过将属性改为 None-No Memory/Protected 删除存在的内存映射范围，这也就是说你对此内存区域既不能读也不能写。
- **重置（Reset）** 将内存映射范围重设为默认数值。

3. 单击 **Done** 保存你的设置。

调试器允许你进入覆盖了原内存的新内存区域。新内存假定设为有效状态的话，被覆盖的区域的属性也会随之改变。

在你定义一个内存映射关系后，你可能想修改它的读/写属性，你可以通过在原来起始

地址和大小基础上定义一个新的内存映射并单击 **Add** 按钮来实现。调试器将会将原来的属性修改为新定义的属性。

5.1.3.3 用 GEL 定义内存映射（Defining Memory Map with GEL）

内存映射也可以通过通用扩展语言（general extension language (GEL) 嵌入函数方式来定义。GEL提供一整套内存映射函数。你可以通过在一个GEL文件中加入内存映射函数并在开始中运行这个函数很容易的建立一个内存映射。（参看5.5.1）。

当你第一次调用Code Composer Studio IDE，内存映射会关闭。你可以进入任何内存区域而不受内存映射的限制。如果你调用将可选择GEL文件名作为指定参数的Code Composer Studio，GEL文件会自动下载。如果这个文件包括GEL函数StartUp()，这个文件中的GEL函数将被执行。你可以根据环境在文件中指定GEL映射函数来自动定义内存映射。使用下面的GEL函数来定义你的内存映射：

Function	Description
GEL_MapAdd()	Memory map add
GEL_MapDelete()	Memory map delete
GEL_MapOn()	Enable memory map
GEL_MapOff()	Disable memory map
GEL_MapReset()	Reset memory map

表5-1 GEL功能

GEL_MapAdd()函数定义了一个有效的内存范围并指定了它的读/写属性。下面是一个例子，利用GEL文件定义两个长度为**0xF000**且可读写的内存映射。

```
StartUp()
{
GEL_MapOn();
GEL_MapReset();
GEL_MapAdd(0, 0, 0xF000, 1, 1);
GEL_MapAdd(0, 1, 0xF000, 1, 1);
}
```

当你设置好你的内存映射，选择**Option->Memory Map**去查看内存映射情况。

5.1.4 引脚连接（Pin Connect）

引脚连接工具使你可以指定外部中断发生的时间间隔。

仿真外部中断：

1. 创建一个数据文件指定中断间隔。
2. 从工具菜单中选择**Pin Connect**。
3. 选择**Pin name**并单击**Connect**。
4. 下载你的程序。
5. 运行程序。

关于Pin Connect的详细信息，可查阅在线帮助中的Pin Connect主题：

Help→Contents→Debugging→Analysis Tools for Debugging→Pin Connect。

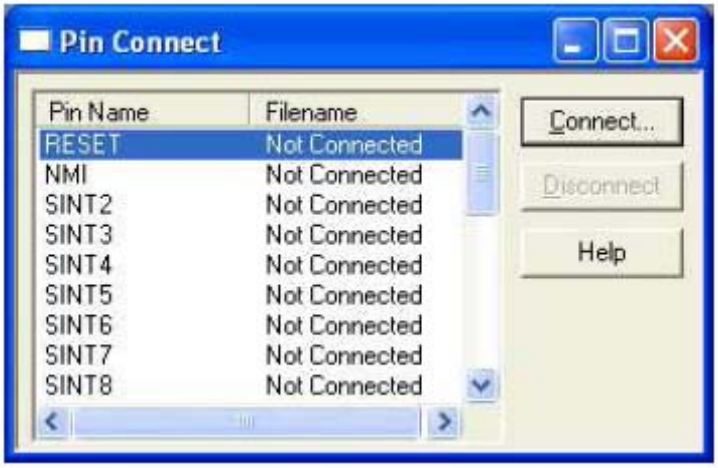

图5-3 引脚连接工具

5.1.5 端口连接（Port Connect）

你可以利用端口连接工具通过内存地址来访问文件。通过连接到内存（端口 ）地址，你可以将数据导出或写入文件。

连接一个内存（端口）地址到一个数据文件，按照以下步骤：

1. 从工具菜单中选择**Port Connect**，显示Port Connect窗口并开始Port Connect工具。

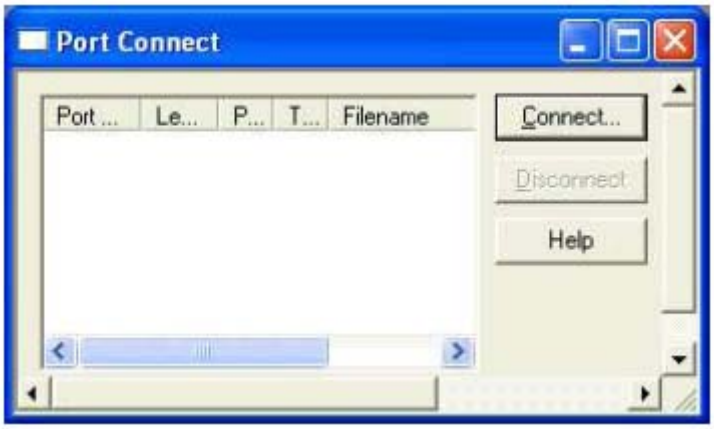

图5-4 端口连接工具

2. 点击**Connect**按钮打开Connect对话框。

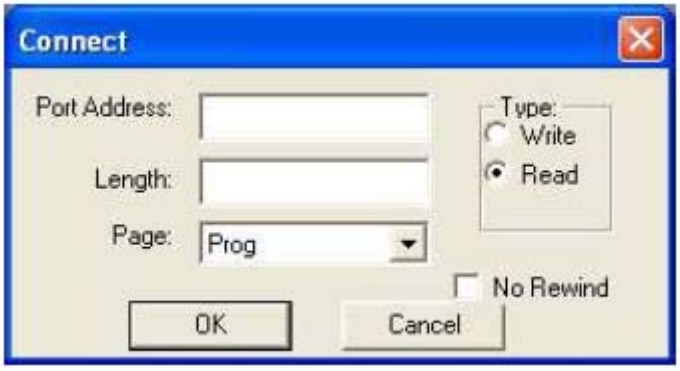

图5-5 端口地址连接

3. 在Port Address区域输入内存地址。这个参数是一个绝对地址，可以是任何C语言表达，C函数名称或者是汇编语言标签。如果你想指定一个hex地址，一定要以0x开头，否则它将被当作十进制地址对待。
4. 在Length区域，输入内存范围的长度。长度可以是任何C语言表达。
5. 在Page区域（仅C5000有），选择地址占据的内存的类型（program or I/O ）。对于程序内存选择Prog，对于I/O则选择I/O。
6. 在Type区域，选择Write还是Read按钮取决于你想将数据写入文件还是读出文件。
7. 点击**OK**显示Open Port File窗口。
8. 选择你想连接的数据文件并点击**Open**。
9. 选择No Rewind特性，防止到达文件结尾end-of-file(EOF)时文件重绕，在EOF后的读访问，数据0xFFFFFFFF被读入并且文件指针保持不变。

文件可通过相关联的内存地址使用汇编语言来进行访问，任何内存地址可以连接到文件。一个最大的单输入输出文件可以连接到一个单内存地址，多个地址可以连接到同一个文件。

关于Port Connect的详细信息，可以参阅在线帮助中的Port Connect主题：

Help→Contents→Debugging→Analysis Tools for Debugging→Port Connect。

5.1.6 程序加载（Program Load）

在执行程序前，必须将由编译程序所产生的 COFF 文件（.out）载入到真实的或仿真的目标芯片中。

将程序的代码的数据分别载入到 COFF 文件中所注释的目标地址中。把字符载入字符表中并保存到主机中的调试器中。字符根据 COFF 文件中的注释载入到程序和数据地址中。载入一个 COFF 文件可以通过选择文件 **File->Load Program** 并使用载入程序对话框来选择所需要的 COFF 文件。

5.1.6.1 只载入字符信息

在调试器环境下工作时（该调试器无法或不需要载入目标代码，例如代码在只读存储器的时候），只载入字符信息是十分有效的。

载入字符信息可以通过选择文件 **File->Load Symbols->Load Symbols Only from the main menu** 并使用载入字符对话框来选择所需要的 COFF 文件。

调试器会将主机上保存的字符表中的先前已调用过的字符删除。然后字符表将载入字符文件中的字符。字符根据字符文件中的注释载入到代码和数据地址中。这个命令不会修改存储器或者设置程序入口点。

你也可以指定代码的偏移量和数据的偏移量，这样调试器就可以在指定的字符文件中应用每一个字符。例如，假如你有一个可执行的字符文件，这个字符文件包含初始化地址为 0x100 的代码地址以及初始化地址为 0x1000 的数据地址。然而当程序载入目标芯片中时，相应的代码初始地址为 0x500100 以及相应的数据初始地址为 0x501000。

指定代码和数据的偏移量，可以先选择 **File ->Load symbols->Load Symols with offsets from the main menu** 并使用载入字符对话框来选择所需要的 COFF 文件。一旦一个 COFF 文件选定后，一个新增的载入带偏移量的字符对话框将会显示出来，并可以在对话框中输入实际的代码和数据的初始地址。

图 5-6 数据地址

5.1.6.2 只增加字符信息

字符信息也可以附加到现存的字符表中。这个命令与载入字符命有所不同，因为这个命令不会在载入新的字符前清除现存的字符表。

增添字符信息的步骤如下：**File→Load Symbols→Load Symbols with Offsets**，或者不带偏移量的 **File→Load Symbols→Load Symbols Only,**，与上述载入字符的步骤相似。

5.2 基础调试（Basic Debugging）

在CCS集成系统中有几个组件在进行基础调试中是非常重要的。以下图表显示出一系列在CCS中调试时使用的图标。如果这些图标在工具栏中无法显示，请选择**View→Debug Toolbars→ASM/Source Stepping**。在这个调试工具栏选项表中，你可以看到许多调试工具的列表，并且你可以将想要的调试工具设置为可视。在菜单栏中，可视的工具名字旁有个校验标记。

图 5-7 运行和调试所用到的一些工具栏图标

5.2.1 运行/单步调试（Running/Stepping）

5.2.1.1 运行

运行一个程序，首先在集成环境菜单中的调试项目中选择合适的命令。如果目标控制工具栏是可视的，运行图标会显示在左边的垂直工具栏中。如果这些图标并没有显示出来，可以选择 **View→Debug Toolbars→Target Control**。

可以运用这些命令来运行程序：

- **主程序(Main)** 可以通过先择 **Debug→Go Main**，来开始对主程序的调试。这个执行命令将会执行主程序函数。
- **运行(Run)** 在执行停止后，可以通过点击 **Run** 按钮来继续运行程序。
- **运行到光标处(Run to Cursor)** 如果想要程序运行到一个指定的位置，可以先把光标移到该位置，然后按下这个按键。
- **驱动(Animate)** 这个执行命令将一直运行程序直到运行到断点处。在断点处，执行停止并且将更新所有与任何试探点（probe point）没有联系的窗口。试探点（probe point）停止执行并更新所有图表及与之有关的窗口，然后继续运行程序。按下这个按键就可以驱动（Animate）执行程序。还可以在选项菜单中选择用户化来修改

驱动（Animate）的速度。

- **停止(Halt)** 最后，可以在任意时候按下停止按键来终止程序执行。

5.2.1.2 单步调试

只有在执行程序的时候源程序和汇编程序的单步调试才可以使用。源程序的单步调试是通过单步执行源程序编辑器中所显示的代码行，而汇编程序的单步调试是通过单步执行反汇编窗口中显示的指令行。通过 **View→Mixed Source/ASM** 来切换源程序/汇编程序混合模式，可以同时查看源代码的汇编代码。

执行一个单步调试命令，先在工具栏中选择合适单步调试图标。另一种方法是先选择 **Debug→Assembly/Source Stepping**（然后选择合适的命令）。

单步调试共有三种：

- 单步调试或者只执行一个表达式然后就终止程序执行。
- 跳过整个函数的执行然后当函数返回时终止程序。
- 跳出执行当前的子程序并返回到调用函数入口。当返回到调用函数入口时，程序就终止了。

5.2.1.3 使用 PDM 来进行多处理器广播命令

当使用并行调试管理器（PDM）时，所有的运行/单步调试命令都会广播到当前处理器组中所有目标处理器。如果设备驱动支持同步操作，以下每个命令都可以同时同步到每个处理器中。

- 使用锁定单步调试(Step into) 来单步调试所有还没准备好运行的处理器。
- 使用跳过单步调试(Step over)来跳过所有还没准备好运行的处理器。
- 如果所有处理器都在子程序中，可以使用跳出执行(Step out)来跳出执行所有还没准备好运行的处理器中的命令。
- 运行命令发出全局运行命令运行所有还没准备好运行的处理器。
- 停止命令(Halt) 将同时终止所有运行的处理器程序。
- 驱动命令（Animate）开始驱动所有还没准备好运行的处理器。
- 在从当前 PC 位置中执行载入程序之前，自由运行(Run free)将禁用所有断点包括所有试探点。

5.2.2 断点（Breakpoints）

对于任意调试器，断点都是十分重要的组成部分。断点会停止程序的执行。当程序停止时，可以检查程序的状态，检查或修改变量，检查调用堆栈等等。断点可以设置在编辑窗口中人任意一行源代码中或者设置在反汇编窗口的任意一个反汇编指令上。在设置完一个断点后，可以启用断点也可以禁用断点。

如果一个断点设置在一行源程序代码行上，必须有一行反汇编代码行与之相对应。在打开编译器优化选项后，许多源程序代码行就不再允许设置断点了。可以在编辑窗口中的混合模式下查看可以设置断点的代码行。

注意：

CCS 会在源程序窗口中重新定位断点到一个有效代码行上并设置断点图标在该代码行的边缘空白处。如果一行允许设置断点的代码行无法设置断点，系统将会以消息窗形式自动报错。

注意：

只要程序执行到任意一个试探点时，CCS 将会终止目标程序。当执行停止时，将会自动更新任何与试探点有关的窗口或输出设备。因此，如果使用试探点，目标应用程序也许就不能实现实时运行的效果。这个开发阶段，就要测试下所使用的算法。然后再使用 RTDX 和 DSP/BIOS 来分析实时效果。

5.2.2.1 软件断点

可以在任意一个反汇编窗口或者含有 C/C++源代码的文档窗口设置断点。只要断点设置的位置合适，对于断点的数量便没有限制。软件断点通过改变目标程序使之在需要的位置增加一条断点指令。

设置软件断点的方法：

1. 在一个文档窗口或者反汇编窗口，移动指针到你想要设置断点的那一行。
2. 当你在文档窗口设置断点时，只需在选定行的前面的页边空白处迅速双击即可。若是在反汇编窗口，则只需在选定行双击。

在选定行的页边空白处的一个实心红点即为断点标志，它表示在所需要的位置已经设定了一个断点。

我们也可以使用切换断点命令和切换断点按钮来迅速的设置和清楚断点。

1. 在一个文档窗口或者反汇编窗口，移动指针到你想要设置断点的那一行。
2. 点击鼠标右键并选择切换断点，或者在软件工具栏中点击切换断点标志按钮。

5.2.2.2 硬件断点

硬件断点与软件断点不同的是它们并不改变目标程序，而是利用芯片上可以利用的硬件资源。硬件中断的用途是在只读存储器或者存储进程中设置断点，而不是获取指令。我们可以在特定的存储器读、存储器写或者存储器读写中设置断点。存储器存取断点并不会在源程序或者存储器窗口中显示出来。你可以使用的硬件断点的数量取决于你所采用的 DSP 型号。

硬件断点也有计数的功能，它决定了在断点产生前，该处指令已经运行的次数。如果计数为 1，则每次到该位置则产生断点。但是，在仿真目标上不能实现硬件断点。

设置硬件断点的方法：

1. 选择 **Debug->Breakpoints**。在选择断点这一栏后，便会出现 **Break/Probe Points** 对话框。
2. 在 **Breakpoint type** 一栏，选择 **H/W Break** 作为指令获取断点，或者在特定位置选择 ***Break on <bus> <Read|Write|R/W>***作为存储读取断点。
3. 在程序或存储器中你想设置断点的某个位置，按以下方法中的一种操作：

   － 对于一个绝对地址，你可以输入任意 C 语言中的表达方式：C 的函数名或者一个标志符号。

   － 输入断点的位置基于你的 C 源文件。当你不知道 C 指令在可执行文件中的位置时，这就很方便了。在基于 C 源文件的位置输入的格式是：文件名 第几行 总行数。

4. 在计数这一栏，输入断点产生前，该处指令需要运行的次数。如果计数设为 1，则每次到该位置便产生断点。
5. 点击添加按钮可以产生一个新的断点。这样便可创造一个新的断点并对其激活。
6. 点击 **OK**。

5.2.3 探针点（Probe Points）

5.2.3.1 探针点的作用

探针点从你的个人电脑上的文件中读取数据。它们有助于算法的开发。你可以用它们来达到以下目的：

- 将主机的一个文件中的输入数据转移到目标的缓冲器中以备算法使用。
- 将目标的缓冲器中的输出数据转移到主机的一个文件中以备分析。
- 运用数据升级一个窗口（例如图表）。

5.2.3.2 探针点与断点之间的不同

探针点和断点都会使目标停止并完成某些动作。然而，它们在以下几方面不同：

- 探针点使目标立即停止，执行一个单一的行动然后便恢复目标的执行。
- 断点会使 CPU 一直停止直到手动恢复执行。并且断点会使所有打开的窗口保持最新。
- 试探点允许文件的自动输入和输出，而断点则不允许。

5.2.3.3 运用试探点转移电脑文件中的数据到一个目标

在这一部分，我们将展示如何运用一个试探点将电脑文件中的内容转移到目标中，并用于数据测试。当试探点到来时，目标程序也会运用断点使所有的窗口保持最新。

1. 选择 **File->Load Program**。选择一个文件，并打开。
2. 在 Project View 中选择要打开的 C 程序源文件并双击打开。
3. 将光标移动到主函数中你想要添加试探点的那一行。
4. 点击 **Toggle Software Probe Point** 按钮。
5. 在文件菜单中，选择 File I/O.对话框，然后你便可选择输入或输出文件。

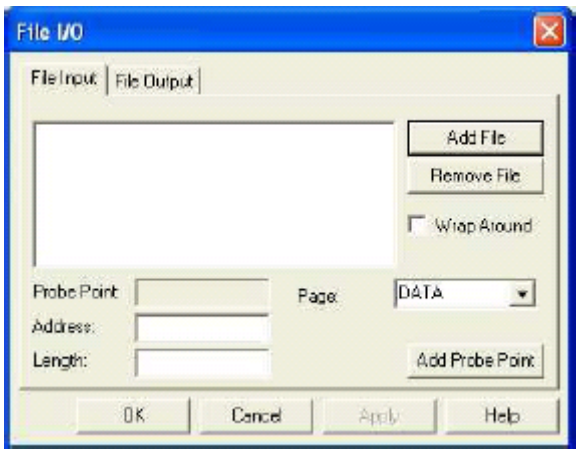

图 5-8 文件输入/输出对话框

6. 在文件输入这一栏，点击 **Add** 添加文件。
7. 浏览到你的文件夹，选择一个数据文件并打开，便会出现一个关于该数据文件的控制窗口。当你运行这个程序时，在数据文件内部可以利用这个窗口来使程序启动、停止、重运行或者快进。

图 5-9 数据文件控制

8. 在 File I/O 中，改变地址和长度值。并且，将 Wrap Around 这一栏打上勾。地址这一栏表明把文件中的数据放到哪里。Length 这一栏表明当试探点到来时，已经从数据文件中读取了多少个样本。Wrap Around 选项使得当到达文件的末尾时又开始从文件的开头读取数据，如此便可把数据文件当作一段连续的数据流。

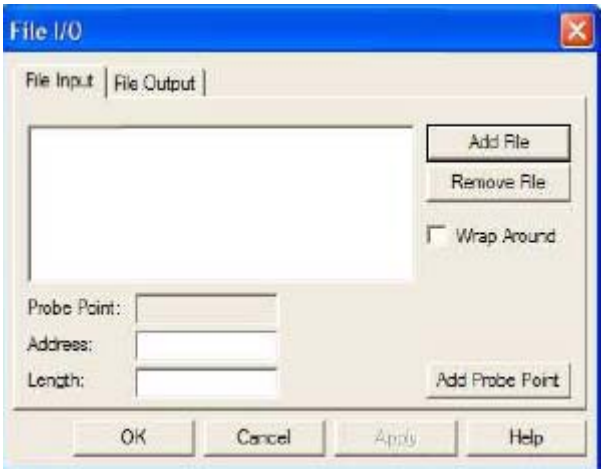

图 5-10 添加你的文件

9. 点击Add Probe Point以显示Break/Probe Points对话框中的Probe Points 一栏。

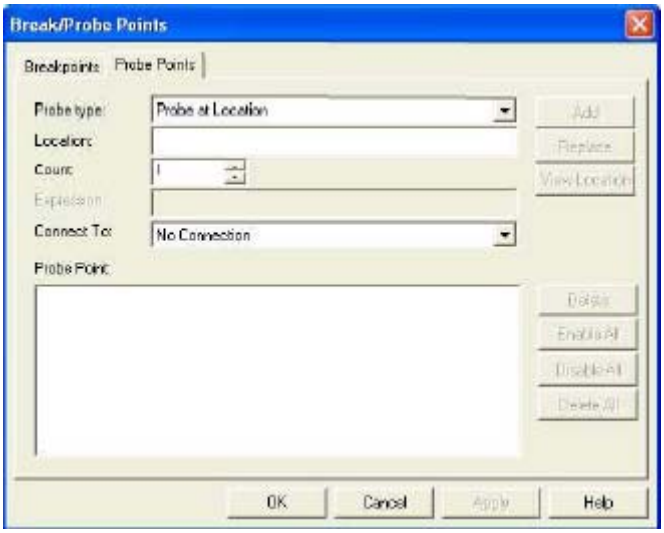

图 5-11 试探点标签

10. 在Probe point list中，选择你之前创造的试探点。
11. 在Connect To field中，点击向下的箭头并从列表中选择一个数据文件。
12. 点击**Replace**。那么试探点列表便会改变并显示该点已经关联到正弦数据文件。
13. 点击**OK**。那么文件输入/输出对话框便会显示该文件已经连接到一个试探点。
14. 点击**OK**，从而选择文件输入/输出对话框。

5.2.4 观察窗口（Watch Window）

5.2.4.1 使用观察窗口来跟踪一个变量的值

调试一个程序的时候，能了解变量的值在程序执行时是如何变化的，这是非常有用的。观察窗口允许用户观察局部变量和全局变量还有 C/C+ +表达式的值。如果想知道观察窗口的详细资料，参看关于观察窗口主题的在线帮助：Help→Contents→Debugging→Viewing Debug Information→Watch Window。

打开观察窗口：

1. 选择 **View→Watch Window**，或者点击观察工具栏上的观察窗口图标按钮。观察到窗口 包含两个统计表：Watch Locals 和 Watch 1。

   -在Watch Locals统计表中，调试器自动显示当前正在执行函数的局部变量的名称、值的大小、类型和基的选择（Radix option）。

   -在 Watch 1 统计表中，调试器显示局部变量、全局变量和用户指定表达式的名称、值的大小、类型和基的选择（Radix option）。
2. 选择 **File→Load Program**。
3. 双击在**Project View** 中的filename.c 文件（filename为文件名）。
4. 把光标移动到允许打断点的行。
5. 点击 **Toggle Breakpoint** 工具栏按钮或按下F9。 被选择的页面空白处会显示出断点已经建立好了（红色图标）。
6. 选择 **View->Watch Window**。 在窗口的右下角会出现一个单独的区域，在运行过程中这个区域显示的就是被观察的变量的值。默认情况下， 显示的是Watch

Locals统计表，显示内容是执行过的函数的局部变量的值。

7. 如果不是在主函数，选择**Debug->Go Main**。
8. 选择**Debug->Run**,，或按下F5, 或按下运行图标。观察窗口会更新局部的值。

图5-12 .Watch Locals统计表

9. 选择 Watch 1 统计表.
10. 在Name column点击Expression图标并且输入需要观察的变量的名称。
11. 点击窗口的空白处可以保存所做的改动。值会立即显示出来，就像下面这个例子。

图 5-13. 指定一个要观察的变量

12. 点击 **Step Over** 工具栏按钮或按下 F10 来跳过对要观察的变量的调用。
    除了观察一个简单的变量的值之外,用户还可以观察一个结构体中的元素的值。

5.2.4.2 利用观察窗口来观察一个结构体中元素的值

用观察窗口来观察一个结构体中元素的值：

1. 选择Watch 1 统计表。
2. 点击**Name** 栏中表达式图表并且输入需要观察的表达式的名称。

3. 点击窗口的空白处用来保存所做的改动。
4. 一旦点击“+”标记，该目录会展开并列出结构体中所有的元素以及他们对应的值。（所示的连接地址可能会有所不同。 ）

图 5-14. 观察元素的值

5. 双击在结构提中任意一个元素的值，就可以对这个值进行编辑。
6. 改变这个变量的值。

需要注意的是在观察窗口中的值如果发生了改变，这个值的颜色也会变成红色用来表示它已经被手动的进行了修改。

5.2.5 内存窗口（Memory Window）

内存窗口允许用户观察由指定地址开始的存储单元中的内容。 用户可以通过选项对内存窗口的显示进行格式化，也可以编辑被选择的存储单元的内容。

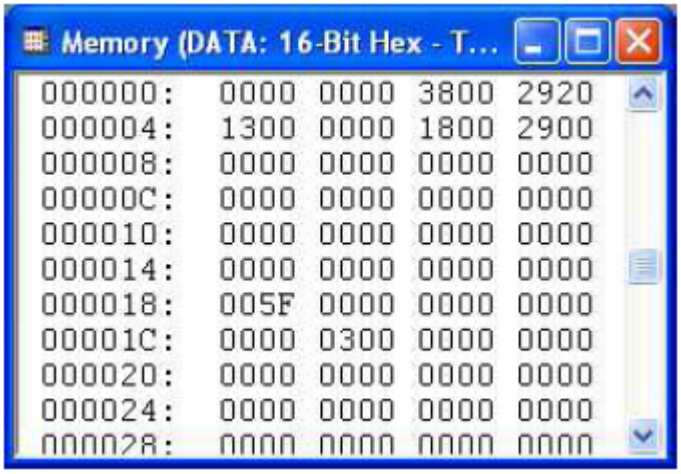

图 5-15. 内存窗口

可以在内存窗口选项对话框中定义内存窗口不同的特性。

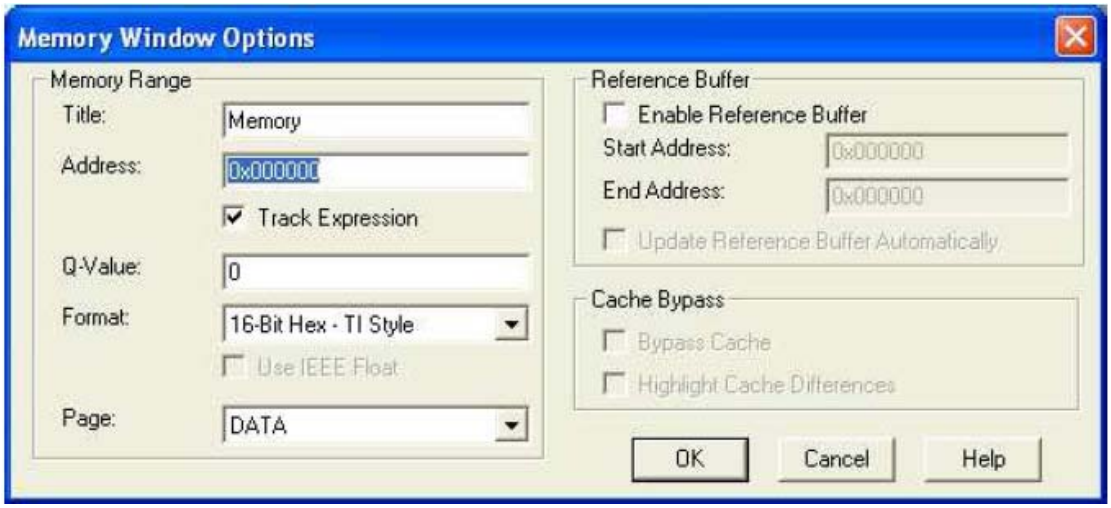

图 5-16. 内存窗口选项

该对话框提供了以下内存窗口选项：

- **Title 标题** 为内存窗口输入一个有意义的名字。当打开内存窗口时，这个名字会显示在标题栏上。当有多个内存窗口打开时，标题会起到十分重要的作用。
- **Address 地址** 输入需要观察的存储单元的起始地址。
- **Track Expression 跟踪表达式** 点击这个选项会使内存窗口自动地重新评估并且改变它基于与起始地址相关联的表达式的起始地址。当目标停止，重新启动，或改变其符号时，它将重新评估并重新配置本身。举例来说，如果在 SP 中打开一个内存窗口，当单步执行代码、加载一个新的程序或者修正寄存器本身时，内存窗口将不断地重新配置。
- **Q 值** 用户可以用 Q 值来显示整数。这个值将整数值表示成更精确的二进制值。小数点被插到二进制值中，最低有效位(LSB) 产生的偏移量由 Q 值决定，如下所示：

  New_integer_value = integer / (2^(Q value))

  xx 的Q值表示一个有符号的2次方余整数（ 2s complement integer） ，这个数的小数点由最低有效位 (LSB) 转移到xx位置 。
- **Format 格式** 从下拉菜单中选择内存显示的格式。关于不同格式的更多信息可以参看在线帮助。
- **Enable Reference Buffer 参考缓冲器有效** 为指定的内存区域保存一个快照，可以用来为后面的比较做准备。.
- **Start Address 起始地址** 输入想要保存在参考缓冲器中的存储单元的起始地址。这个区域只有当“激活参考缓冲器”选择以后才会激活。
- **End Address 终止地址** 输入想要保存在参考缓冲器中的存储单元的终止地址。这个区域只有当选“激活参考缓冲器”择以后才会激活。
- **Update Reference Buffer Automatically 自动更新参考缓冲器** 选择这个复选框可以自动地用指定地址区域的当前内存内容覆盖参考缓冲器的内容。如果选择该选项，只要内存窗口更新，参考缓冲器就会更新（例如，当选择了更新窗口，打了断点，或者对目标的执行被终止时）。如果该复选框没有被选中，参考缓冲器的内容就不会改变。这个选项只有当选择了“Enable Reference Buffer”以后才会激活。
- **Bypass Cache 旁路高速缓存** 该选项使得内存总是从物理内存中读取内存内容。

通常情况下，如果内存内容在高速缓存中，内存的返回值会是从高速缓存中得到的值，而不是从物理内存中得到的。如果这个选项被激活，CCS（Code Composer Studio）将忽略或者绕过高速缓存的内容。

- **Highlight Cache Differences 突出高速缓存的差异** 当高速缓存的值和物理值不一致时，这个选项突出强调了的存储单元的值。也会用色彩来加强突出高速缓存的差异。选择 **Option→Customize→Color** 并且选择在Screen Element 下拉框中的Cache Bypass Differences 选项。

参看关于寄存器窗口的在线帮助部分可以了解跟详细的信息。

5.2.6 寄存器窗口（Register Window）

用户可以在寄存器窗口观察并编辑选中的不同寄存器的内容。

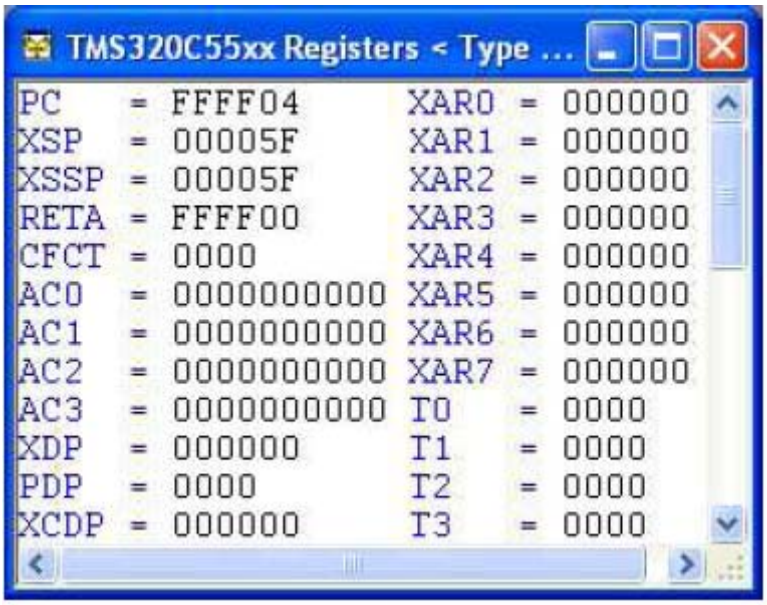

图 5-17. 寄存器窗口

要访问寄存器窗口，选择**View→Registers**并且选择需要观察/编辑的寄存器组。 如果要访问寄存器的内容，选择**Edit→Edit Register**，或者从寄存器窗口双击一个寄存器，或者右击一个寄存器并选择Edit Register（编辑寄存器）。

Figure 5-18. 编辑寄存器的值

5.2.7 反汇编模式/混合模式（ Disassembly/Mixed Mode）

5.2.7.1 返汇编模式 Disassembly Mode

当你加载程序到实际的或模拟的目标板时，调试器会自动打开一个反汇编窗口（disassembly window）.

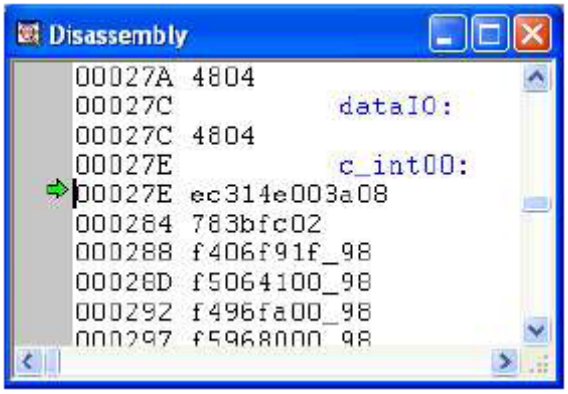

图 5-19 分解窗口

反汇编窗口显示反汇编的指令和符号信息以供调试需要。反汇编与汇编过程相反，并允许内存的内容以汇编语言代码的形式显示。符号信息包括符号和代表目标板上地址或值的文字数字式字符串。

如果以单步命令运行程序，程序计数器将跟随这些指令向下执行。

5.2.7.2 混合模式 Mixed Mode

除了在分解窗口里查看分解的指令，调试器允许以 C 源代码和分解代码交叉显示方式察看。点击 **View→Mixed Source/ASM,** 或在源文件窗口中右击，根据你目前的模式选择 Mixed Mode 或 Source Mode。

5.2.8 调用堆栈（Call Stack）

可用调用堆栈窗口察看使程序运行到目前位置的函数调用。

要显示调用堆栈窗口：

1. 选择 **View→Call Stack**，或单击调试工具条上的 View Stack 按钮。

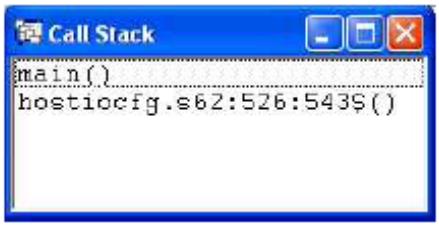

图 5-20 调用堆栈窗口

2. 双击调用堆栈窗口中列出的一个函数。包含该函数的源代码将显示在一个文档窗口中。指针置于函数内的当前行。一旦在调用堆栈窗口中选择了某个函数，你可以查看该函数内的局部变量。

调用堆栈功能只在 C 程序中使用。调用函数由扫描（walking through）运行时栈里的帧指针链决定。程序必须有一个堆栈段和一个主函数；否则，调用堆栈是不能显示 C 源代码的。另外还要注意，调用堆栈窗口只显示输出的 100 行，100 行以外的行都忽略不显示。

5.2.9 符号浏览器（Symbol Brower）

符号浏览窗口（Figure 5-21）能为加载的 COFF 输出文件（*.out）显示 5 个标签窗口（Tabbed windows）：

- 所有相关的文件
- 函数
- 全局变量
- 类型
- 标记（labels）

每个标签窗口包含代表各种符号的节点。节点前的加号（+）表示此节点可被进一步展开。可单击加号展开节点。展开的节点前有一个负号（-），可单击负号隐藏该节点的内容。要打开符号浏览窗口，选择 **View→Symbol Browser**。

图 5-21 符号浏览器窗口

5.2.10 命令窗口（Command Window）

命令窗口使你可以依据 TI 调试器的命令语法对调试器指定命令。

许多的命令接受 C 表达式语法为参数。这使得指令相对较小却仍足够强大。由于评测某些类型的 C 表达式会影响既有的值，你可以使用相同的命令来显示或改变某个值。

要打开命令窗口，选择 **Tool→Command Window**。

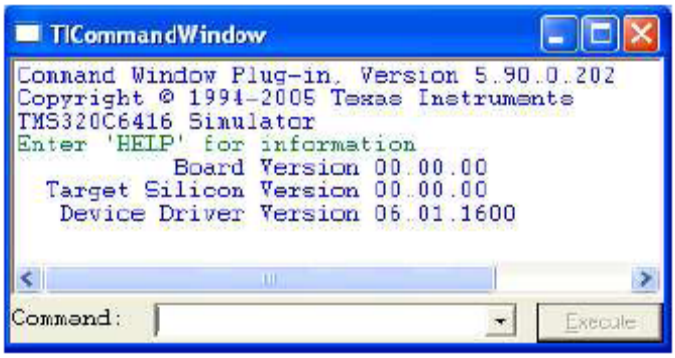

图 5-22 命令窗口

更多关于命令窗口的信息，请参看在线帮助提供的 Command Window 项。

5.3 高级的调试特征（Advanced Debugging Features）

5.3.1 高级事件触发（Advanced Event Triggering）

高级事件触发(AET)使用两个工具：**事件分析(Event Analysis)**和**事件序列器(Event Sequencer)**来简化硬件分析。

事件分析采用了很简单的界面来配置一般的硬件调试任务。你可以用一个上下文菜单和一个拖放动作就很方便的设置断点、激活点(action points)和计数器。可以通过工具菜单或在源文件中右击进入事件分析(Event Analysis)。

事件序列器在你的目标程序里寻找状态标志(conditions)，当探测到状态标志时就启动指定的动作(specific actions)。CPU暂停时，你可以定义状态标志和动作，然后再次运行你的目标程序。序列程序会寻找指定的状态，并执行被请求的动作。

5.3.1.1 事件分析 （Event Analysis）

使用事件分析可以执行下列任务：

- 设置断点

—硬件断点
—带计数的硬件断点
—连锁断点
—全局硬件断点

- 设置动态点/观察点

—数据动态点
—程序动态点
—观察点
—带数据的观察点

- 设置计数器

—数据存取计数器(Data access counter)
—剖析计数器(Profile counter)
—看门狗计时器

——般计数器

- 其它

—标记基准点(Benchmark to here)

—硬件仿真引脚配置(Emulation pin configuration)

更多关于事件分析工具的信息，请访问在线帮助提供的事件分析主题。

为了使用事件分析工具来配置任务，必须为集成有片上分析特性(on-chip analysis features)的目标处理机配置CCS IDE。你可以通过从工具菜单选择或在源文件中右击来使用事件分析。一旦你配置好一个任务，它就被激活了，当你在目标板上运行代码时，它会执行分析任务。更多关于如何激活和禁用一个已经配置好的任务，请访问高级事件控制的在线帮助。

1. 选择**Tools→Advanced Event Triggering→Event Analysis**查看事件分析窗口。

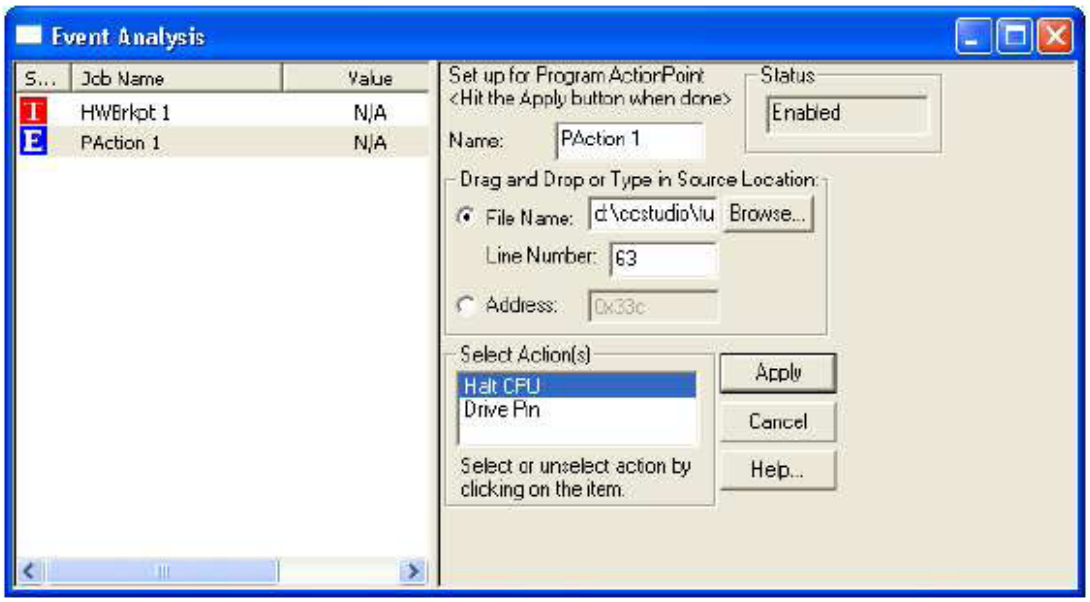

图5-23事件分析窗口

2. 在事件分析窗口里右击，选择**Event Triggering->Job Type->Job**。任务的菜单被动态建立并取决于目标的配置。若你的目标板上不支持某个任务，这个任务将显示灰色。
3. 在Job对话框里输入信息。
4. 点击**Apply**保存你的任务。

5.3.1.2 事件序列器（ Event Sequencer）

事件序列器在你的目标程序里寻找状态标志(conditions)，当探测到状态标志时就启动指定的动作。CPU暂停时，你可以定义状态标志和动作，然后再次运行你的目标程序。序列程序会寻找指定的状态，并执行被请求的动作。

为了使用事件序列器，必须为集成有片上分析特性(on-chip analysis features)的目标处理机配置CCS IDE。你可以通过从工具菜单选择来使用事件序列器。一旦你创建好一个事件序列器程序，它就被激活了，当你在目标板上运行代码时，它会执行分析。更多关于创建一

个事件序列器程序，请访问高级事件控制的在线帮助。
激活事件序列器：

1. 选择**Tools→Advanced Event Triggering→Event Sequencer**。事件序列器将显示如图

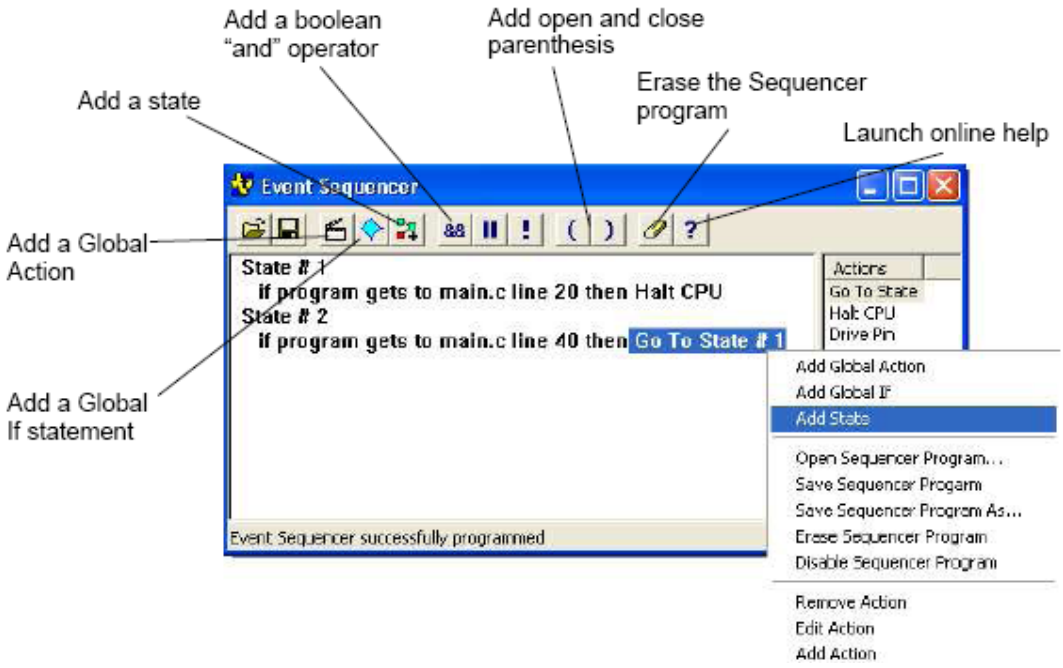

图5-24 事件序列器

2. 在 Event Sequencer 窗口中右击或者使用 Event Sequencer 工具栏的按钮创建一个次序器程序。

5.4、实时调试（Real-Time Debugging）

传统的调试方法需要（停止模式）需要程序员完全停止他们的系统，这将停止所有的进程而且阻止了中断的响应。只要系统/应用程序没有实时要求，停止模式可以用于调试。然而为了更好的测量你应用程序的实时系统行为，代码编译集成环境平台(CCS IDE)为你提供了几种方式，包括实时模式(5.4.1 节)、强制实时模式(5.4.2)、实时数据交换模式(5.4.3)。

5.4.1 实时模式（Real-Time Mode）

实时模式允许当目标系统暂停在后台程序里时，可以响应前台代码里时间紧急的中断。时间紧急的中断是一类在后台程序暂停时必须响应的中断。例如：时间紧急的中断可能是马达控制器和高速的定时器的中断。你可以在多处挂起程序，这样允许你在响应一个紧急时间中断时打断去响应别的中断。

激活实时模式调试：

1. 你需要配置中断和设置断点，为实时模式调试做好准备，看网上的关于实时调试帮助以获得更多的信息。选择 **Debug→Real-time Mode**，控制框的底部状态条此时显示温和实时模式。

2. 通过选择 **View→Real-Time Refresh Options**，配置实时刷新选项。第一个选项将确定观察窗多久更新一次。选择全局连续刷新选择框，将连续的刷新所有的窗口，包括内存、图表和观察窗。如只要连续更新部分窗口，不选择全局框，在观察窗列表里选择相应的框即可。
3. 点击 **OK** 关闭对话框。
4. 选择 **View→Registers→Core Registers** 来打开主寄存器，调试中断激活寄存器(DIER)在列表里可见，DIER 代表了一个单一中断、一类特殊设置的中断、所有的通过中断激活寄存器(IER)选择的作为实时模式的中断，DIER 反映了特殊的 IER。
5. 右击寄存器，选择编辑寄存器。
6. 输入新的寄存器值以表示实时中断。
7. 选择 **Done** 关闭寄存器编辑对话框。
8. CCS IDE 已经配置好实时模式调试。

5.4.2 强制实时模式（Rude Real-Time Mode）

高优先级的中断，或者其它的代码段可以是非常时间紧急性，执行花费的周期数要最小。这意味着调试行为（包括执行控制和访问寄存器和内存）可能被禁止，在一些代码段或特殊的机器状态下。在默认的实时模式下，控制器放弃了特权来运行温和模式。这样调试行为将等待一个合适的时间延迟，不会闯入调试敏感的窗口。

然而，假如不能获得调试 action-senstive 的窗口，调试命令（包括执行控制和访问寄存器和内存）将要失败。为了使调试能再次获得控制权，必须把实时调试从温和模式改为强制模式。在强制实时模式下，特权的拥有允许调试行为忽略任何阻止调试进入的手段，并且没有延迟的成功执行。你也可以在进入强制实时模式之前不调试响应时间迫切的代码。

为激活强制实时模式，执行下面中的一个：

- 当调试命令失败时，从显示窗口中选择执行强制模式。
- 当实时模式激活后，在调试菜单里选择激活强制实时模式。

当强制实时模式激活后，主程序窗口底部的状态栏显示强制实时模式。要取消强制模式，在调试菜单里撤销激活强制模式按钮即可。状态栏此刻显示温和实时模式。

假如激活了强制实时模式，且暂停了 CPU，可能会发生这种即时调试被阻止了，CPU 还是处于停止状态。这可能发生在一些时间紧急的中断服务程序里。这阻止了 CPU 在合时的时间里完成中断服务程序，因为在对断点响应之前，CPU 不能运行。为避免此问题，必须撤消强制模式返回至温和模式。

通过选择 **Debugging→Real Time Debugging** 的网上帮助，了解更多的该主题的信息。

5.4.3 实时数据交换（RTDX）

DSP/BIOS 实时分析工具使用实时数据交换（RTDX）连接以实时获得或监测目标系统的数据。你可以利用 RTDX 连接和 RTDX API 库，创建自己的同 DSP 目标系统通信的用户接口。

RTDX 在没有干扰目标系统的程序下，在主机和目标系统之间传送数据。这个双向的通信通路提供了主机和正在运行程序的目标系统之间数据采集和发送。从目标系统上采集的数据可以被分析和显示在主机上。可以在不停止目标系统的程序下，用主机调整其参数。RDTX 也使主机能够提供对目标系统程序和算法的数据仿真。

RTDX 包括主机部分和目标系统部分，一个小的 RTDX 软件库运行在目标系统程序里，目标系统程序调用该库的 API 函数来发送或接收数据。该库使用基于扫描的仿真器，通过 JTAG 口接收或发送数据至主机。目标系统程序执行的同时，数据实时发送到主机。

在主机平台上，一个 RTDX 主机库和 CCS IDE 连接起来，数据可视化和分析工具通过 COM APIs 和 RTDX 通信，获得目标系统数据或发送数据到 DSP 程序。

主机库支持两种接受数据的方式：连续和非连续。连续模式下，数据仅被 RTDX 主机库简单缓存起来，没有写入日志文件。连续模式用于开发人员想从目标系统程序里连续获得并显示数据，而不需要保存数据到日志文件里的场合。非连续模式下，数据写入到主机的日志文件里，该模式用于开发人员想捕获一定量的数据且保存起来的场合。

要获得使用 RTDX 更多的细节，看网上帮助。

5.4.3.1、RTDX 数据流

RTDX 在主机和目标系统之间建立两通道的数据管道。数据管道由如下图所示的硬件和软件的结合。

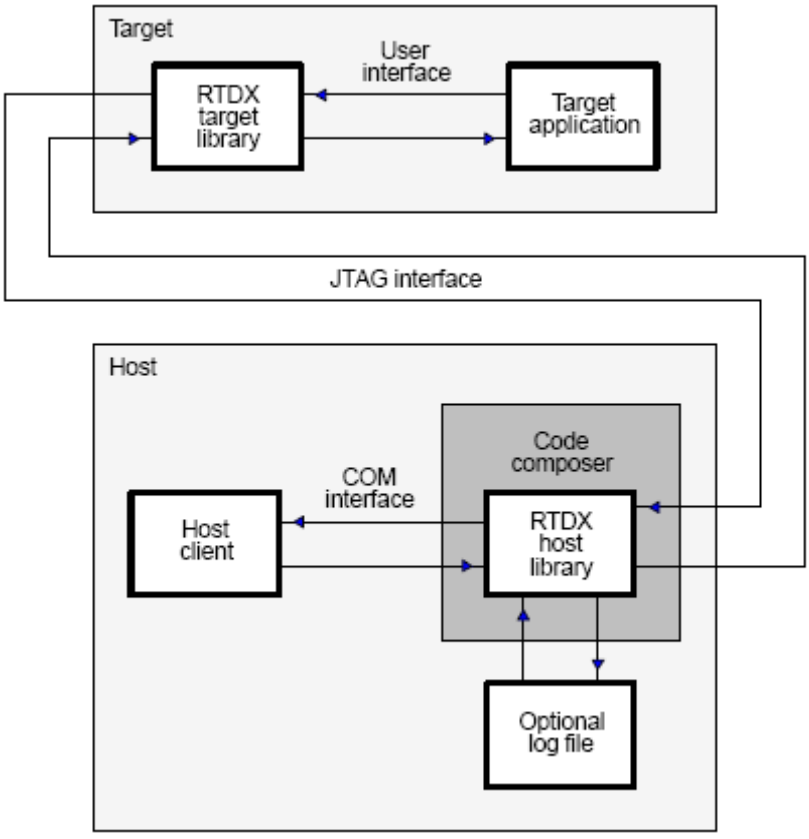

图 5-25 RTDX 数据流程

5.4.3.2、图形化配置 RTDX

RTDX 工具允许图形化配置 RTDX，建立 RTDX 通道，开始 RTDX 诊断。这些工具允许你在传输数据时候增强 RTDX 函数。

RTDX 拥有三个菜单选项：诊断控制器、配置控制器、通道视图控制器。

诊断控制器：RTDX 提供了 RTDX 诊断控制器来诊断 RTDX 在系统里正常工作与否。

诊断器测试基本的目标系统到主机，主机到目标系统数据传输的函数。

选择**Tools→RTDX→Diagnostics Control**，打开RTDX诊断控制器。这些测试只有在RTDX激活才可用。

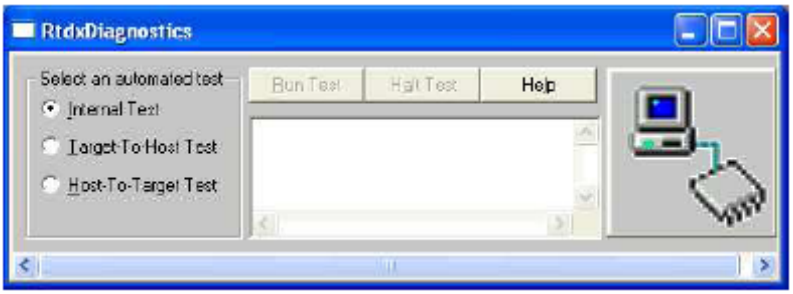

图 5-26 RTDX 诊断窗口

配置控制器：下面是 RTDX 控制器窗口，它允许你完成下面事：

- 查看当前的 RTDX 配置
- 激活或关闭 RTDX
- 打开 RTDX 配置属性页面，重新配置 RTDX，选择点配置设置属性。

选择 **Tools→RTDX→Configuration Control**.，打开配置控制器。

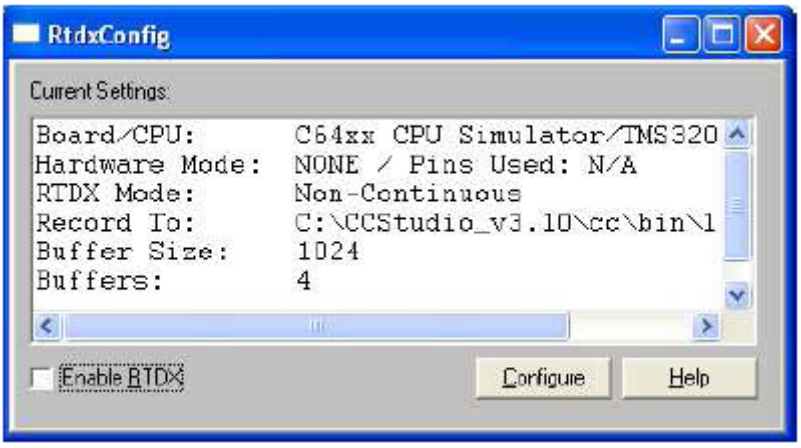

图 5-27 RTDX 配置窗口

通道视图控制器：RTDX 通道视图控制器是一种主动 X 控制器（Active X Control），它自动检测目标系统声明的通道，并将其加入到可视的列表里。RTDX 通道视图控制器允许你完成下面事：

- 从可视的列表里删除或增加目标系统声明的通道。
- 激活或关闭该列表里的通道。

选择 **Tools→RTDX→Channel Viewer Control**，在 CCS IDE 里打开 RTDX 通道视图控制器，通道视图控制器窗口如下：

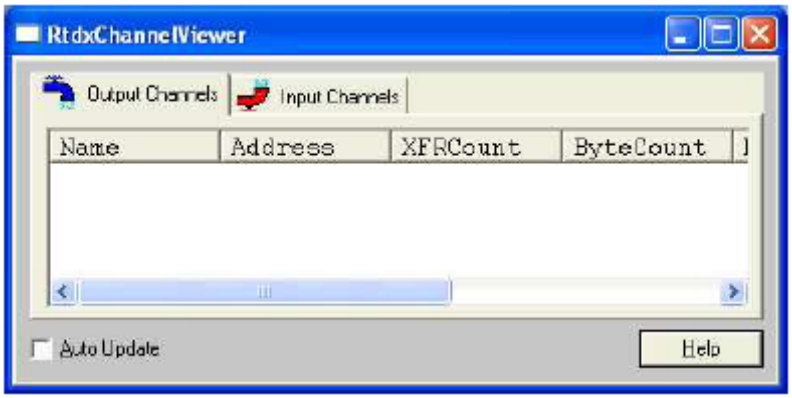

图 5-28 RTDX 通道浏览窗口

点击输入和输出通道标签显示相应通道的列表。输入和输出通道窗口都可以允许你查看、删除、重加载通道。

选择 Auto-Update，可以自动更新所有通道的数据，即使你没有刷新显示窗口。假如你不使用自动刷新，手动右击标签，选择刷新，这样也可以刷新所有通道的数据。

注意：要使 RTDX 通道视图控制器接受一个特殊的扩展通道信息，必须要 RTDX 主机将该通道打开。

5.4.3.3、发送一个整数至主机

RTDX 的一个基本函数是发送整数到主机。下面步骤提供了目标系统到主机、主机到目标系统的发送数据过程的概述。传送不同类型的数据所需的命令和细节请参阅网上 RTDX 帮助。

从目标系统程序发送数据到主机：

1. 准备目标系统程序以获取实时数据，通过插入特殊的 RTDX 参数在程序里，来实时传送数据到主机。虽然准备目标系统程序的过程对于所有类型的数据传输都一样，但不同的数据类型所需不同的函数。所以，发送一个整数到主机需要添加特定的函数仅适用传送一个整数的，而不能发送一个整型数组。
2. 准备主机程序接受数据，为每一个所需的通道初始化相应的 RTDX 目标，打开特定目标的通道，调用所需的函数。
3. 打开 CCS IDE
4. 加载目标程序到 TI 处理器。
5. 在 **Tools—>RTDX—>Configuration Control** 选择激活 RTDX。
6. 运行目标系统程序，实时捕获数据，发送到 RTDX 主机库。
7. 运行主机程序处理数据。
8. 欲获的更多使用 RTDX 细节，参考网上 RTDX 帮助。

5.4.3.4、接受主机数据

主机程序通过往目标系统写数据来传送数据。从主机传送过来的数据首先缓存在 RTDX 的主机库里。在来自目标系统的数据请求到来之前，数据保存在 RTDX 主机库里。一旦 RTDX 主机库有足够多的数据满足请求时，在不干扰目标系统程序运行下将数据写入目标系统。

缓存的状态返回至缓存状态变量里，正值表示在目标系统请求到达之前，RTDX 主机库里缓存的字节数。负值表示目标系统请求已到达，RTDX 库还缺少的字节数。

从主机发送数据到目标系统程序：

1. 准备好目标系统程序的接受数据，通过写一个简单的从主机读数据的 RTDX 目标程序来完成。
2. 准备好主机发送数据程序，为每个需要的通道初始化一个 RTDX 目标，打开特定目标的通道，调用其它所需函数。
3. 打开 CCS IDE。
4. 加载目标程序到 TI 处理器。
5. 在 **Tools→RTDX→Configuration Control** 里选择激活 RTDX。
6. 运行目标系统程序，实时捕获数据，发送到 RTDX 主机库。
7. 运行主机程序处理数据。
8. 欲获的更多使用 RTDX 细节，参考网上 RTDX 帮助。

5.5 自动控制（Automation for Debug）

5.5.1 使用通用扩展语言 (GEL)

像前面讲述的一样，在 CCS 中，GEL 脚本可以用来创建常用的 GEL 菜单和自动运行步骤。4.6.1 节讲述了怎样使用 GEL 的构造函数来自动运行各种工程管理步骤。另外还有许多的 GEL 构造函数，可以用来自动调试，例如设定断点、向观察窗口增加变量、开始运行、停止和设定输入输出文件 I/O。

5.5.2 脚本效用 (Scripting Utility for Debug)

脚本效用(4.6.2 节)也有许多可以自动运行调试步骤的命令。通过在线帮助可以查看关于脚本语言效用的更多的信息。

5.6 重置选项（Reset Options）

有时有必要使用 CCS IDE 中的附加命令，实现目标芯片或者仿真器的重置。这些重置命令的适用性取决于 IDE 与目标芯片的联系。要得到更多的关于目标芯片连接的信息，可以查看 3.1.3 节。

5.6.1 目标芯片重置（Target Reset）

目标芯片的重置把寄存器的内容初始化为上电状态，同时中断程序的执行。如果目标板没有响应这条命令，而且用户正在使用以内核为基础的设备驱动，这时 CPU 核可能已经损坏，这种情况下，需要重载 CPU 内核。

模拟器会初始化所有寄存器的内容到上电状态，这可以参照目标芯片仿真的说明。

要重置目标芯片，选择 debug 的 Reset CPU 选项。

注意：必须把目标芯片做好连接，以确保 Reset CPU 选项可用。

5.6.2 仿真重置（Emulator Reset）

在硬件重置工作之前，有些处理器需要把处理器调整到功能性的运行状态。在这种情况下，把处理器返回到功能性状态的唯一方法就是重置仿真器。一个仿真器的重置需要把 TRST 管脚激活，使设备进入到功能性的运行模式。

在 CCS 没有连接到目标芯片的时候，重置仿真器的选项可用。重置仿真器时，选择 **Debug->Reset Emulator** 的选项。在运行仿真器重置时，硬件处于空转状态，此时可以手动进行目标芯片的硬件重置，方法是：按 reset 按钮，或者选择 **Debug-> Reset CPU** 选项。但这种方法不适合于 ARM 设备。

第六章　分析/调整

为了创建一个高效的应用程序，用户根据自己的目标需要考虑性能、功耗、代码大小、成本等因素。

应用程序代码分析是对影响程序效率因素的数据进行收集和解释的过程。应用程序的调整就是修改代码以此提高程序效率。DSP 开发人员可以尽可能的分析和调整应用程序，以此来满足用户、应用区域和硬件所要求的效率。

CCS IDE 提供各种各样的工具来帮助开发者分析和调整应用程序。

6.1 程序代码分析

CCS IDE提供的各种分析工具来获得重要的数据并将数据呈现给开发人员。

6.1.1 数据可视化（Data Visualization）

CCS IDE可以通过各种方法把程序处理到的数据画出来，包括时间/频率，星座图，眼图和图像。

选择**View-> Graph**，可以得到这些图，然后选择需要的图。这时候可以在图属性（graph property）的对话框中指定图的属性选项。下面的例子展示了一个单时刻(时间/频率)图表属性对话框。

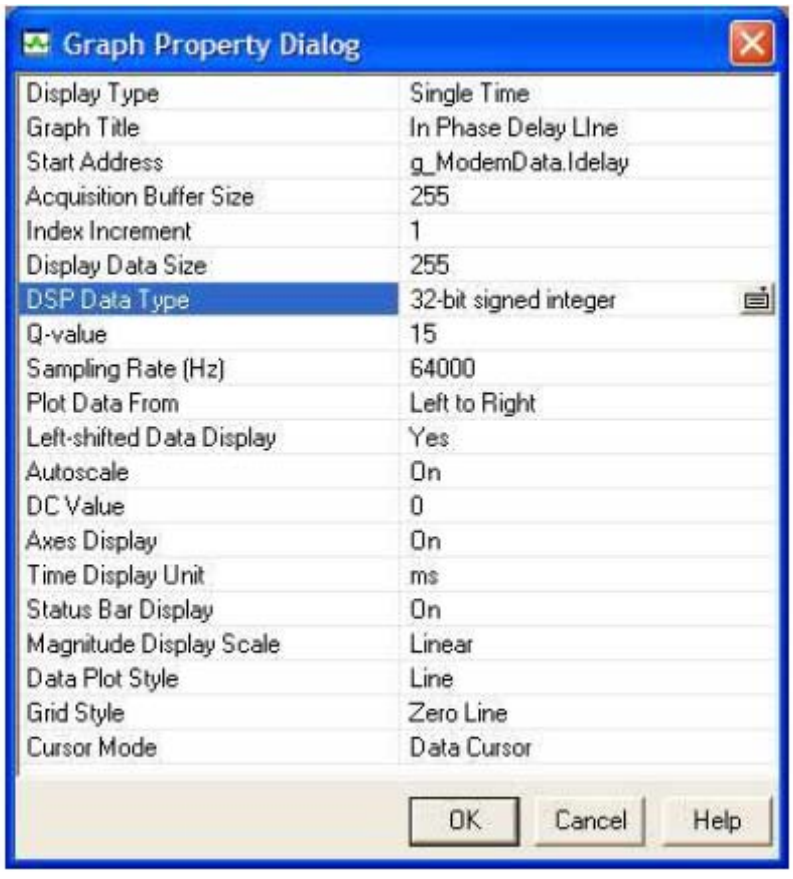

图6-1 图表道具样式对话框

一旦属性被设定了，选择**OK**按钮，开启图窗口，就会画出指定的数据点。

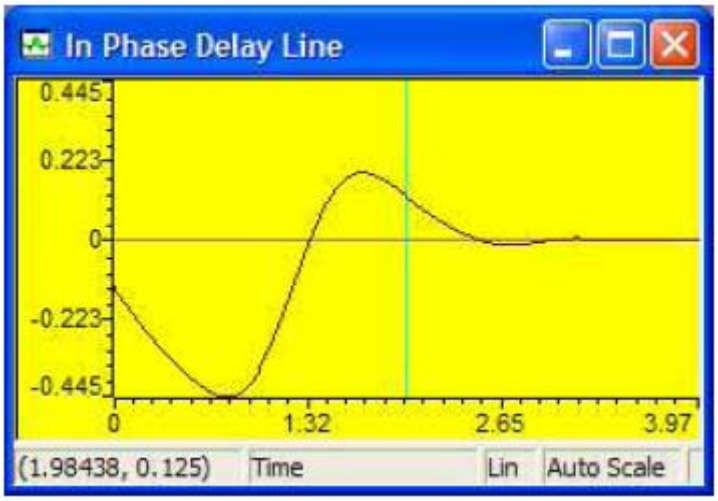

图6-2 图表示例

可以查看在线帮助和指南来获取更详细的信息。

6.1.2 模拟器分析（Simulator Analysis）

模拟器分析工具可以报告特定的系统事件，这样用户就可以准确的监视和衡量程序的性能。

模拟器分析选项有：

- 激活/关闭分析功能
- 计算选定事件的发生次数
- 当选定事件发生时停止运行
- 清除计数或者暂停事件
- 创建日志文件
- 重设事件计数器

可以使用模拟分析工具如下：

1. 下载程序
2. 开始分析工具。选择**Tools->Simulator Analysis**
3. 如果还没有启动，选择**Simulator Analysis-> Enable Analysis**
4. 运行或者单步运行用户程序
5. 对分析工具的输出进行分析

要获得关于模拟分析工具的详细信息，可以通过在线帮助得到模拟分析的更多信息。

6.1.3 仿真分析（Emulator Analysis）

仿真分析工具允许用户对事件、硬件断点进行设定、监视、计数等操作。

为了启动仿真分析工具，首先下载程序，然后从菜单栏中为你的设备选择

Tools->Emulator Analysis选项。

仿真分析窗口包括以下栏的信息：

- **Event 事件** 事件名
- **Type 类型** 事件是打断事件的还是记数事件
- **Count 计数** 在程序停止前事件发生的次数
- **Break Address 打断地址** 打断事件发生的地址
- **Routine 常规** 打断事件发生的规律

注意:

当你正在用剖析时钟时，不能使用分析特征（analysis feature）功能

要得到仿真分析工具的更多的信息，可以从在线帮助的仿真分析主题中得到。

6.1.4 DSP/BIOS 实时分析(RTA)工具

DSP/BIOS实时分析(RTA)工具的特征，如图6-3所示，提供给用户对于应用程序的可见性，这是通过对一个正在运行的DSP应用程序进行设置探针、跟踪、监控来实现的。这些工具与前面提到过的调试器使用了相同的物理连接（JTAG），同时在目标芯片和主机间使用这种连接作为一种低速通信连接。

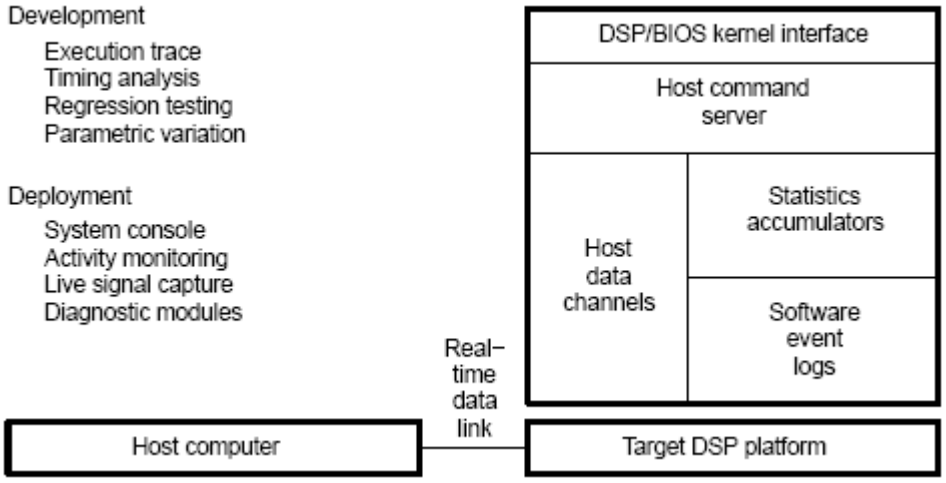

图6-3 实时捕获和分析

DSP/BIOS RTA要求在目标系统中有DSP/BIOS内核。另外为了给应用程序提供即（run-time）服务，DSP/BIOS核可以在通过物理连接与主机通信方面提供帮助。通过围绕DSP/BIOS APIs构造应用函数，静态创建目标，使用者可以自动配置目标芯片，来获得实时信息并上传给主机，这些信息驱动CCS的可视的分析工具。在芯片程序控制下，辅助的APIs和目标允许外部数据被获得。从主机工具的角度出发，DSP/BIOS可以为实时程序分析提供可扩展能力。

DSP/BIOS实时分析工具可以在DSP/BIOS工具条中访问(图6-4)。

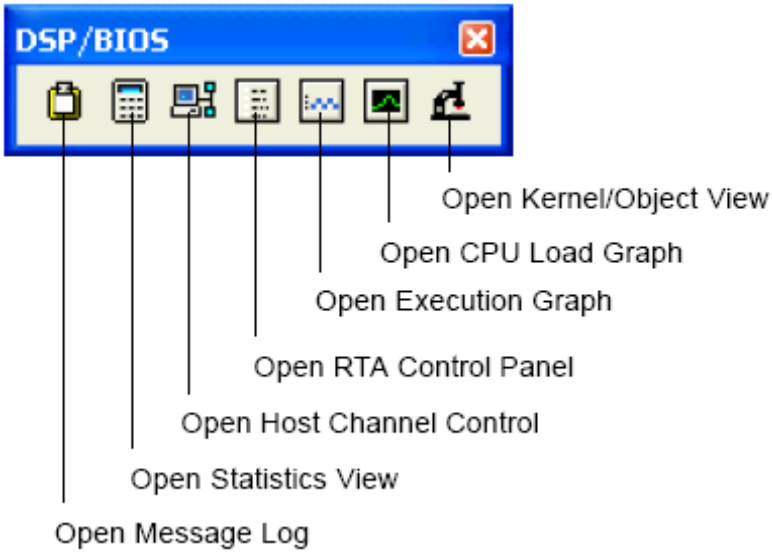

图6-4 DSP/BIOS RTA 图标

下面介绍工具条上每个按钮

- **消息日志（Message Log）** 通过独立的实时序列来显示事件的时序结果，这些结果被写入内核日志模块。这对于跟踪程序的控制流是很有用的。目标板程序通过下面两种途径记录事件：
  ——明确地，是通过DSP/BIOS API函数来调用。例如，可以采用LOG_printf（&trace,“hello world!”），这里的“trace”是日志模块的名称。
  ——含蓄地，当线程变成准备、被调度和中止时由潜在的内核程序来记录。例如这些结果将显示在程序执行细节（Execution Graph Details）的日志事件里。
  你可以将日志输出到一个文件，通过右击“Message Log”窗口，选择属性页面。
- **状态统计显示（Statistics View）** 显示了对于内核累加器简明的统计数据，反映了动态的程序元素包括单个计数器和时变的数据值，独立线程已消耗的处理时间。目标板程序状态统计可以明确地通过DSP/BIOS API函数来调用，或是隐含地当线程在执行或在做I/O操作时有内核程序来记录。
- **主机通道控制（Host Channel Control）** 显示在程序中定义的主机通道。用户可以使用这个窗口来指定与这些通道相连接的数据文件，还可以在一个通道开始数据传输时监视传输数据量。绑定内核I/O模块和主机文件可以为目标板程序提供标准的数据流，来准确地测试算法。其它由内核I/O模块处理的实时目标板数据流，可以被主机文件实时第捕获用来分析。
- **实时分析控制面板（RTA Control Panel）** 控制在目标板程序里的实时跟踪和状态统计。这有效地使开发者控制实时程序的可视程度。默认情况下，所有类型的跟踪都是被允许的。你必须选中“Global host enable”选项来激活所有跟踪类型。用户程序可以在这个窗口改变设定。实时分析控制面板会检查任何程序性变化，以在其属性页面设置的频率。在属性页面里，用户可以改变实时分析工具的数刷新率，比如程序执行状态图“Execution Graph”。
- **程序执行状态图（Execution Graph）** 实时显示线程的执行情况。通过程序执行状态图用户可以看到线程被执行的时序。深蓝线表示了正在运行的线程，就是正在使用CPU的线程。更多关于图表上不同线的信息，可以通过右击“Execution Graph”

窗口选择图例来获得。如果用户在“Message Log”窗口显示程序执行细节，用户可以双击在程序执行状态图一段线，以文本形式来看事件的细节。用户也可以右击“Execution Graph”窗口来选择属性页来隐藏一些线程。

- **CPU负荷图（CPU Load Graph）** 显示了目标板CPU在处理时负荷的图。当前的CPU负荷在图的左下角，而CPU所达到的最高负荷则出现在右下角。CPU负荷定义为没有花在低优先级任务上的时间，而通常认为在没有其它线程需要运行时，低优先级任务才会运行。所以CPU负荷包括了从目标板到主机传送数据的时间和运行额外的后台任务所需要的所有时间。CPU负荷是整个查询周期内的平均值。查询周期越长，CPU负荷上的短尖峰越不可能显示在图上。打开实时分析控制面板窗口，右击选择属性页（Property Page），在主机刷新选项卡“Host Refresh Rates tab”中通过移动滑动块“Statistics View / CPU Load Graph”设定查询速率。
- **内核/模块查看窗口（Kernel/Object View）** 显示当前配置以及目标板上运行DSP/BIOS模块的当前状态。该工具可同时显示目标板上存在的静态和动态设定的模块。用户可以右击窗口选择“Save Server Data”来保存当前数据。

注意：当在软件开发中使用CCS集成开发平台的标准调试器时，DSP/BIOS的实时分析工具提供了必要的可见性对于正在执行程序的目标板，而调试器只能提供很少。甚至在调试器中断程序和取得目标板控制以后，已经被DSP/BIOS捕获的信息可以提供宝贵的观察对于直到当前断点的事件时序。

DSP/BIOS实时分析工具也可以作为硬件逻辑分析的软件副本。嵌入式DSP/BIOS内核和主机分析工具结合起来，成为一套新的生产测试和现场诊断工具。通过现有的JTAG方针设备在运行的产品系统中，这些工具能与应用程序进行交互。使用DSP/BIOS的代价是很小的，所以这些工具能做现场诊断，使得开放者们能找到并分析那些导致错误的数据。

6.1.5 代码覆盖范围和多事件剖析工具

代码覆盖范围和多事件剖析工具提供了两种独特的功能：

- 代码覆盖范围提供了源代码行覆盖范围的可视化，以帮助开发者建立测试保证合适的代码覆盖范围
- 多事件剖析工具提供了从多个事件的感兴趣部分收集到的功能剖析数据，所有这些都在应用程序的方针运行中。事件包括CPU周期数、被执行的指令、流水线的延迟、高速缓存的命中与未命中等等。这些工具帮助用户确定影响程序执行的可能因素。

查看“Code Coverage and Multi-event Profiler User’s Guide”（SPRU624）得到进一步的信息。

6.2 应用程序代码调整（ACT）

调整的过程开始于在分析阶段结束后。当应用程序代码分析完成时，DSP开发者确认

那些低效的代码。调整过程包括确定低效的代码是否能被改善，设定有效的模块，和修改代码尽力符合这些目标。CCS将多种工具结合成一套有组织的调整方法，就如同为调整分析过程提供了一个单一的方向。

6.2.1 调整面板（Tuning Dashboard）

这个面板是调整过程的一个重点。它提供了编译和运行时该用那种调整工具来剖析数据的建议，它也可以启动每种工具。这个面板是在开发周期中调整阶段时的主要界面。

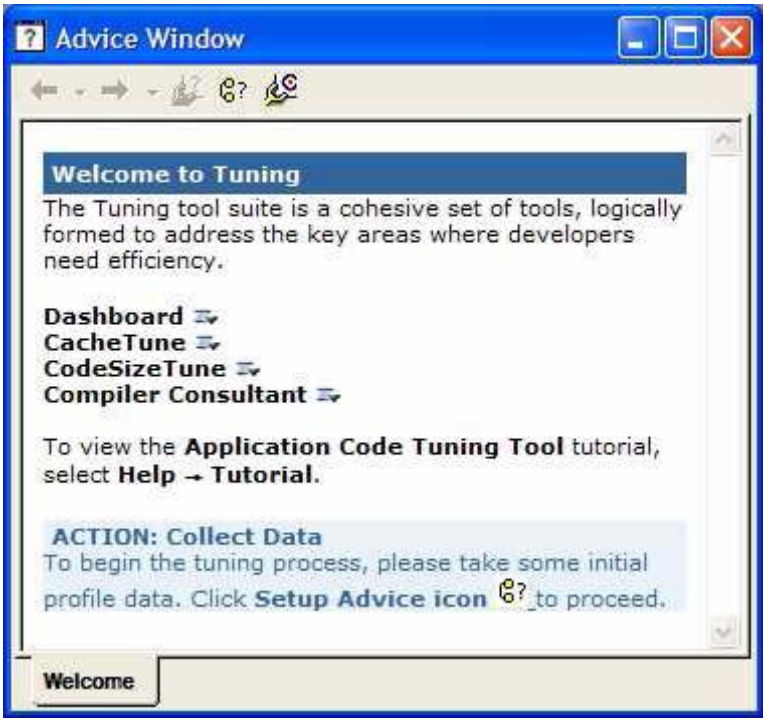

图6-5 优化面板的建议窗口

建议窗口显示了调整的信息，是优化面板的一个组成部分。它在调整的过程中指导用户，解释和详述了合适的步骤，显示了有用的信息，连接了其它工具和文档以及重要的消息。在调整过程的任何节点，当第一次使用这些工具或确定采取合适的手段时，应该考虑建议窗口。

CCS集成开发平台最初开始于调试布局界面。通过点击在工具条上的调整分支图标来打开建议窗口转换到调整布局界面。或者，用户可以在“**Profile→Tuning menu item**”选择“**Advice**” 。建议窗口将出现在CCS IDE屏幕的左侧，并显示欢迎信息。

在调整面板的顶部有一组按钮来用因特网方式浏览建议页面，就象打开建议主页面的一些按钮。单击工具栏的箭头来向前向后浏览建议页面。在中间窗体的下部有一个或多个选项卡。这些选项卡显示不同的建议页面，允许你同时执行多个任务。单击建议窗口底部的选项卡来在已打开的页面中切换。右击建议窗口在菜单中选择“**Close Active Tab**”来关闭活动选项卡。

欢迎页面包括了各种主要调整工具描述的连接。在欢迎页面的底部有一个蓝框来提示调整过程的下一步。这种蓝色的活动控件贯串建议窗口页面的始末。

在浏览建议窗口页面时，红色的警告信息可能会出现。这些信息是有帮助的提示并包含

了宝贵的解决方法对各种问题。

当你打开一个工具时，比如代码尺寸调整“CodeSizeTune”，对于这个工具的建议选项卡会在建议窗口的底部出现。该页面包含了该工具的信息，包括它的功能、用法和怎样应用这个工具来最优化效率。

6.2.1.2 剖析设置（Profile Setup）

如同建议窗口所指出的，CCS IDE在运行前必须确定调整过程所需要的代码单元（code elements）。而这可以用来剖析设置（Profile Setup）完成。“Profile Setup”必须在优化开始时使用，来指定要收集的数据和所要求的代码段。

“Profile Setup”窗口可以用建议窗口（Advice Window）来打开。在欢迎界面底部的蓝框，单击连接打开安装建议界面。第一个活动控件包含了打开“Profile Setup”窗口的连接。“Profile Setup”也可以通过主菜单中的“**Profile-> Setup**”来打开。

活动标签显示了剖析会话将以何种防止收集应用程序的数据。“Ranges”标签指定了所收集数据的范围。函数、循环和任意代码段能被添加到“Ranges”标签来收集优化信息。使用在“Control”标签上的退出点（exit point）来确定CCS何时停止数据采集。用户也可以使用“Control”标签添加暂停或回复采集点来隔离不同代码段。“Custom”标签收集自定义信息，比如高速缓存的命中或CPU空闲周期。

6.2.1.3 目标窗口（Goals Window）

调整一个应用程序需要设定和达到一个有效的目标，所以CCS提供一种方法来记录数字性的目标并跟踪用户的改善。

目标窗口“Goals Window”显示了应用程序数据的汇总，包括代码尺寸和CPU周期数的评估，它可以在代码运行时更新。它也可以比较现在的数据与上一次的数据和设定的目标。选择“Goals”在“**Profile->Tuning**”菜单中，来打开目标窗口。

图6-6 Goals Window

如果应用程序应经被装载并且剖析工具已经安装，目标窗口能获得数据简单地通过运行应用程序。为了记录要优化的模块，单击目标窗口键入目标名称并按回车。如果一个目标达成了，数据将以绿色并加圆括号显示。否则，数据将显示为红色。当应用程序重新运行时，在“Current”栏里的数据将移动到“Previous”栏，它们的差别将显示在“Delta”栏。“Goals Window”允许用户随时保存或查看日志中的数据，通过使用窗口左边的“logging”图标。

6.2.1.4 剖析结果查看器（Profile Viewer）

剖析结果查看器“Profile Viewer”显示了在优化过程中收集到的数据。它包括了一张报

表。每一行对应于在剖析建立窗口范围标签种选择的代码段。每一列包含了每个剖析代码部分的采集数据，就像在剖析建立窗口的活动和自定义标签中选择的那样。

剖析结果查看器提供了一个独立的区域来显示优化过程中采集到的信息。纵列能由不同的数据值来排序。在剖析结果查看器中的数据组可以被保存、恢复、与其它数据组比较。

剖析结果查看器可以精确指出最无效的代码段。比如为确定哪个函数导致了最多的高速缓存延迟，高速缓存延迟数据在查看器中从最大到最小进行排序。然后函数段被剖析来确定哪些代码导致了高速缓存延迟。

在主菜单选择“Profile Viewer”，来打开“**Profile-> Viewer**”。或者，在“Advice Window”中浏览到“Setup”标签。在安装建议页面的底部，单击“Profile Data Viewer”在屏幕底部显示剖析结果查看器。如果在“Profile Setup”窗口中启动了优化过程，运行应用程序将在剖析结果查看器中显示数据。这个查看器可以通过拖曳行和列来由用户定制。数据可以用查看器中的按钮来保存和恢复，也可以通过双击每一列的标题来排序。另外，多个查看其能被同时打开。

6.2.2 编译顾问（Compiler Consultant）

编译顾问工具分析 C/C++源代码，并且提供一些可以提高程序执行性能的特殊建议。编译顾问工具有两种信息：编译时间循环信息（Compile Time Loop Information）和运行时间循环信息（Run Time Loop Information）。编译时间循环信息是被编译者创建的。运行时间循环信息是通过剖析应用程序而整合的数据。每当你编译和创建代码，顾问就会分析代码并且提供针对不同最优化技术的建议用于提高代码效率。你可以选择执行建议重新创建工程，然后用 Profile Viewer 窗口来查看最优化的结果。

当你分析编译顾问信息时，Profile Viewer 中的信息是以不同的列分类的，分类如下：

- If you didn’t profile, sort on Estimated Cycles Per Iteration to see which loops take the most estimated cycles in a single iteration (Compile Time Loop Information).
- If you profiled with the activity Collect Run Time Loop Information, sort on cycle.CPU: Excl. 为了弄清楚哪些循环执行周期最多，忽略系统作用。
- If you profiled with the activity Profile all Functions and Loops for Total Cycles, sort on cycle.Total:Excl. 为了弄清楚哪些循环执行周期最多，考虑系统作用。
- 建议的次数，用以判断哪些循环被建议的次数最多

消耗最多 CPU 周期的将被显示在 Profile Viewer 的最上面一行，通过调整，它的执行效果将被大大提升。然后，你可以每次调整一个循环。双击任何一个循环行的 Advice Types entry, 在建议窗的顾问标签中会生成全面的建议。

在已经应用了建议去修改个别的循环后，应该在 Profile Viewer 窗口中隐藏那一行。这样做可以减少出现在 Profile Viewer 窗口中的信息量。行是可以始终不被隐藏的。

对于更多的信息，在 Application Code Tuning 下的在线帮助中输入 Compiler Consultant 进行查看。

6.2.3 代码尺寸调整（CST）

CodeSizeTune（CST）可以很容易调整应用程序中的代码大小与周期数之间的平衡。使用一系列不同的 profiling 配置，CST 会剖析应用程序，收集各个功能的数据，并且决定编译的最优组合。然后，CST 会产生一个包含了各个特定功能的图表以供选择，你就可以在图表上选择最适合你需要的配置。

以前的 Code Computer Studio 的使用者认为 CST 是 Profile-Based Compiler(PBC)的一种替代。

1. 首先，在 Application Code Tuning 的在线帮助中检查 CodeSizeTune，确保你的程序符合 CST 剖析的标准。
2. CST 会使用一些 profile collection option sets 来创建并剖析你的程序。通过一些剖析收集选项（profile collection options），使用从功能执行上获取的剖析信息，CST 整合收集选项，这些选项包含各种功能水平的编译选项。想要了解更多的信息关于怎样去创建和剖析，在 CodeSizeTune 的在线帮助中查看 Build and Profile。
3. 最优的特定功能选项集将被显示在一个关于代码尺寸和执行性能的二维图表上，这样你就可以选择适合系统需要的一个最优的代码尺寸和运行速度的组合。想要了解更多的关于如何选择需要的收集选项设置，在代码尺寸调整的在线帮助中查看 Select Desired Speed 和 Code Size。
4. 最后，你会讲你所选择的收集选项设置（collection option sets）保存到 Code Computer Studio 工程上。在 CodeSizeTune 的在线帮助中查看 Save Settings 和 Close。

建议窗口将指引你完成以上步骤。当你使用 CodeSizeTune，建议窗口中的 CodeSizeTune 标签将会自动显示。在线帮助中可以查看关于 CodeSizeTune Advice Window 的更多信息。

图 6-7 代码尺寸调整建议

6.2.4 高速缓冲存储器调整（Cache Tune）

Cache Tune 工具提供了一个图表，它使整个高速缓存通路清晰可见。这一工具对于加强非最优化的高速缓存存储器用法十分有效。使用这一工具，可以使高速缓冲存储器效率最优化，从而减少存储子系统中所消耗的周期，并且提高整个程序的高速缓存效率。

所有的存储通路是通过不同颜色编码而分类的。各种各样的滤波器，摄全景，摄像移动特征使其可以迅速的 drill-down to 观察特定的区域。这种高速缓存通路的即时视觉扫描可

以迅速辨别问题区域，例如冲突、容量、强制性错误。

Tuning-> Cache Tune，显示最近的高速缓存踪迹。为了观察高速缓存踪迹，在 Profile Setup 中有各种各样的选项需要被选中，通过在线帮助查看更多信息。有三种类型的高速缓存踪迹文件：

- 程序高速缓存踪迹
- 数据高速缓存踪迹
- 交叉高速缓存踪迹

数据高速缓存踪迹标签以默认值显示。如果没有高速缓存踪迹被采集到，图表是空的。

当此工具被运行时，可以通过打开一个保存过的数据集查看其它高速缓存数据文件。

通过点击 **Open dataset** 按钮，按它的热键，或者点击上下文菜单中的加载数据集项，可以打开数据集。

想要了解关于高速缓存分析工具的更全面的信息，请查看 TI 网上的高速缓存分析用户指引（SPRU575）。

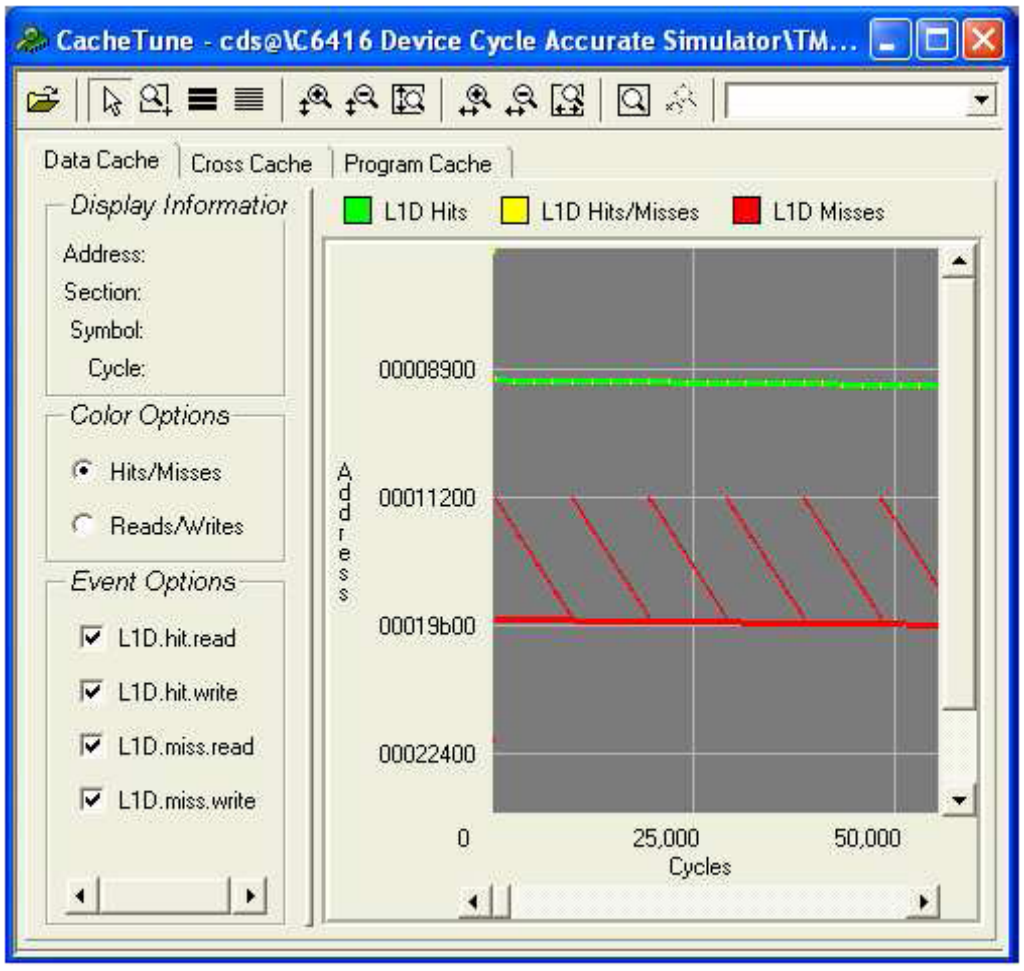

图 6-8 高速缓存分析工具

第七章　其它工具，帮助，小技巧

这一节介绍了怎样制定你的 IDE 安装，怎样对你的安装进行升级，以及怎样找到附带的帮助和文献。

7.1 组件管理器（Component Manager）

提示:
组件管理器是一种高级的工具，它主要用于对你安装进行定义或修改。用这种工具只能对在一个专门的或者多重的安装环境中解决组件中的相互关系。

Code Composer Studio IDE 的多种安装可以共享已经安装的工具。组件管理器为多种安装提供了不同版本工具。

组件管理器窗口列出了所有安装创建工具,德州仪器插件程序工具，以及第三方的插件程序工具。当选择左边的列表中的一个选项时，那么它的属性就会在在右边的属性栏中列出。(如图 7-1)。

在组件管理器的帮助下你能对特殊 Code Composer Studio 安装中的工具开启或者关闭。这种功能可以让你创建一种定制的连接工具，包括 IDE 在内。在组件管理器中你同样可以进入更新导航从网站上下载最新的版本的工具。

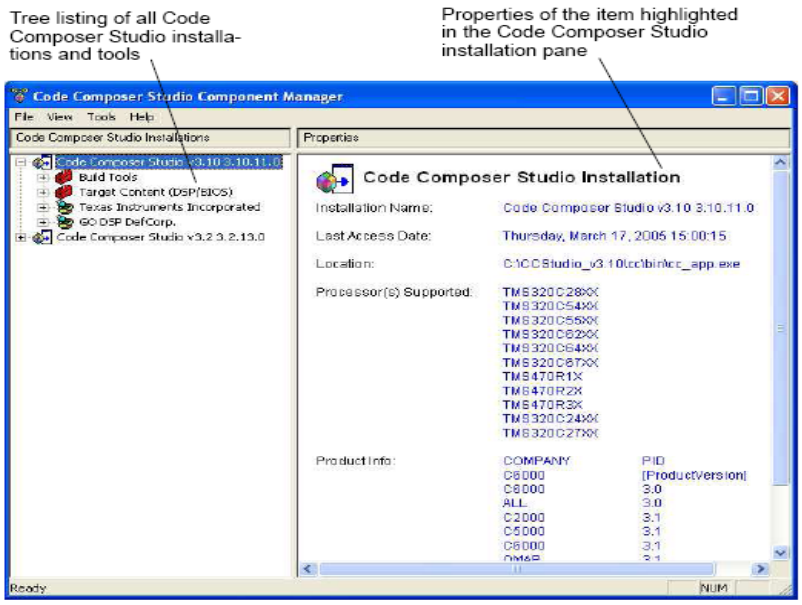

图 7-1 组件管理器

7.1.1 打开组件管理器

打开组件管理器

1. 从 Code Composer Studio IDE 的帮助菜单中，选择关于。这时呈现出 Code Composer Studio 的关于目录栏。
2. 在关于目录栏中，点击组件管理器按钮。这时呈现出组件管理器窗口。

7.1.2 Code Composer Studio IDE 的多种版本

- 下面是一系列关于保持多种 Code Composer Studio IDE 版本的要求以及相关的工具：
- 如果希望拥有多种 Code Composer Studio IDE 版本或相关的工具，你需要把每一种版本安装在不同的目录下
- 如果你把额外的 Code Composer Studio IDE 版本，或者额外版本的工具安装在以前的安装目录下，那么以前的安装将被覆盖。
- 在任一种安装下你不能使用多种版本的相同工具。

7.2 更新导航（Update Advisor）

更新导航可以让你下载不同的 Code Composer Studio IDE 版本以及相关工具。从更新导航里可以进入下载更新的网站。网站可以下载补丁，驱动和祥光工具。

在运用更新导航前你需要连接网络，你计算机里必须有浏览器。

注意：

进入更新网站之前你必须注册 my.TI

7.2.1 下载更新注册

如果你在安装过程中没有注册，那么你可以从 Code Composer Studio 帮助菜单中：帮助→注册进入在线注册系统。

注意：

在你第一次使用更新导航时，你的浏览器也许会显示 my.TI 网页。然后跟着下面的步骤注册。

你必须通过更新导航(Update Advisor)在线注册并获得有效的订阅单。你会得到一个拥有 90 天免费订购服务。在这以后，你必须购买一年的订购服务。

7.2.2 检查工具更新

Code Composer Studio IDE 中，选择帮助 更新导航(Update Advisor) 检查更新 如果你已经在 my.TI 中注册过了并且已经收到了必要的自动注册信息，那么你的浏览器会自动地跳转到有效的更新网页站点。为了查询更新网页站点，更新导航(Update Advisor)会从你的计算机中认证一些信息：

- Code Composer Studio IDE 产品注册码
- Code Composer Studio IDE 安装版本
- 安装产品的文本介绍
- 安装的插件

更新网页站点将列出适合你电脑上的 Code Composer Studio 安装的更新。
你可以下载更新，或者下载并且迅速安装它们。
你也可以设置更新导航来自动检查更新。

7.2.3 自动检查工具更新

你可以在任意时间检查工具更新，或者你可以设置更新导航来自动检查工具更新。

1. 选择 **Help→Update Advisor→Settings**。呈现出网页设置对话栏：

图7-2 网页设置对话栏

2. 按下在 **Enable timed check for update upon startup** 左边的选项框。当这个地方被选定时，更新导航会根据所设定的时间自动的在网上检查更新。
3. 在 **Check for update** 选项栏，指定更新导航在网上检查更新的周期。
4. 按下 **OK** 来保存你的设定，关闭对话框。

7.2.4 卸载更新

任何安装的更新都能卸载，从而恢复以前版本的 Code Composer Studio IDE。注意只有之前的版本可以被恢复。如果已经安装了一个更新版本的工具，然后又安装了第二个更新版本相同的工具，只有第一个更新版本能够被恢复。而原始的版本却不能被恢复，即使你同时卸载掉了第一个和第二个版本的更新。

7.3 附加帮助（Additional Hel）

你可以进入帮助 内容来一步步指导你学习一些主题，进入在线指南来查阅在线帮助

站点，这些网站提供了最新的主题帮助，或者或者在 PDF 中浏览 **user manuals** 来获得一些重要的细节信息或方法。你可以通过帮助　更新导航来获得最新的一些功能特征。

7.3.1 在线帮助

在线帮助与指南，多媒体样品，用户手册，应用报告和网站(www.dspvillage.com)相连接，你可以得到该软件的相关信息。简单的安下帮助按钮然后跟着连接所提供的内容。

对于“上下相关的”帮助，选择与 IDE 相关的部分然后按 F1

7.3.2 在线指南

Code Composer Studio IDE 指南包含了帮助你快速启动该软件的教程。选择帮助　指南来打开 Code Composer Studio IDE 的在线指南工具。