附录　DSP 集成开发环境

本附录介绍 TI 公司的集成开发环境 CCS(Code Composer Studio)。CCS 提供了环境配置、源文件编辑、程序调试、跟踪和分析等工具，可以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。与 TI 提供的早期软件开发工具相比，利用 CCS 能够加快软件开发进程，提高工作效率。

CCS 一般工作在两种模式下：软件仿真器和与硬件开发板相结合的在线编程。前者可以脱离 DSP 芯片，在 PC 机上模拟 DSP 的指令集与工作机制，主要用于前期算法实现和调试。后者实时运行在 DSP 芯片上，可以在线编制和调试应用程序。一般地，一种 CCS 只适用于一种系列的 DSP 芯片，例如 CCS C5000 适用于 C5000 系列 DSP 芯片，包括 C54x 和 C55x。用户只需在 CCS 配置程序中设定 DSP 的类型和开发平台类型即可。

目前TI公司提供的CCS最高版本是2.20版。本章以CCS C5000 v2.20为例，介绍如何利用DSP集成开发环境开发应用程序。文中未详细说明的部分可以通过查阅CCS主菜单Help在线帮助获得，也可参阅TI公司提供的资料SPRU509C《Code Composer Studio Getting Started Guide》。

1　CCS 安装及设置

1.1　系统配置要求

(1) 机器类型：IBM PC 及兼容机。

(2) 操作系统：Microsoft Windows 98/2000、Windows NT(SP6)或 Windows XP Professional and XP Home Edition。

(3) 机器配置要求见表 1，注意当使用硬件开发板时需要主机空余一条 EISA 插槽，以便插入驱动板。

表 1　CCS 安装配置要求

部件	最低配置	推荐配置
内存	64MB	128MB
剩余硬盘空间	600MB	600MB
CPU	Pentium(233MHz)	Pentium III 以上 (500MHz)
显示分辨率	SVGA 800×600	SVGA l024×768
主板插槽	一条空余 ISA 插槽	一条空余 ISA 插槽

1.2 安装 CCS

安装过程包括两个阶段：

(1) 安装 CCS 到系统中。将 CCS 安装光盘放入到光盘驱动器中，此时启动光盘自动运行程序，提示用户是否要安装 CCS2。也可以运行光盘

根目录下的 setup.exe，按照安装提示，一步步完成安装（系统的默认安装目录是 c:\ti，这里我们安装在 d:\ti 下）。同时，建议安装 Acrobat，这是为方便用户阅读 CCS2 自带的帮助文档。如果在 Windows NT 下安装，用户必须要具有系统管理员的权限，安装完成后，在桌面上会有“CCS 2 (‘C5000)”和“Setup CCS 2 (‘C5000)”两个快捷方式图标。分别对应 CCS 应用程序和 CCS 配置程序。

(2) 运行 CCS 配置程序设置驱动程序。如果 CCS 是在硬件目标板上运行，则先要安装目标板驱动卡，然后运行“CCS Setup”配置驱动程序。最后才能执行 CCS。除非用户改变 CCS 应用平台类型。否则只需运行一次 CCS 配置程序。

1.3 “CCS setup”配置程序

“CCS setup”配置程序用来定义 DSP 芯片和目标板类型。双击桌面上的“Setup CCS 2 (‘C5000)”快捷方式图标。弹出对话框如图 1 所示。

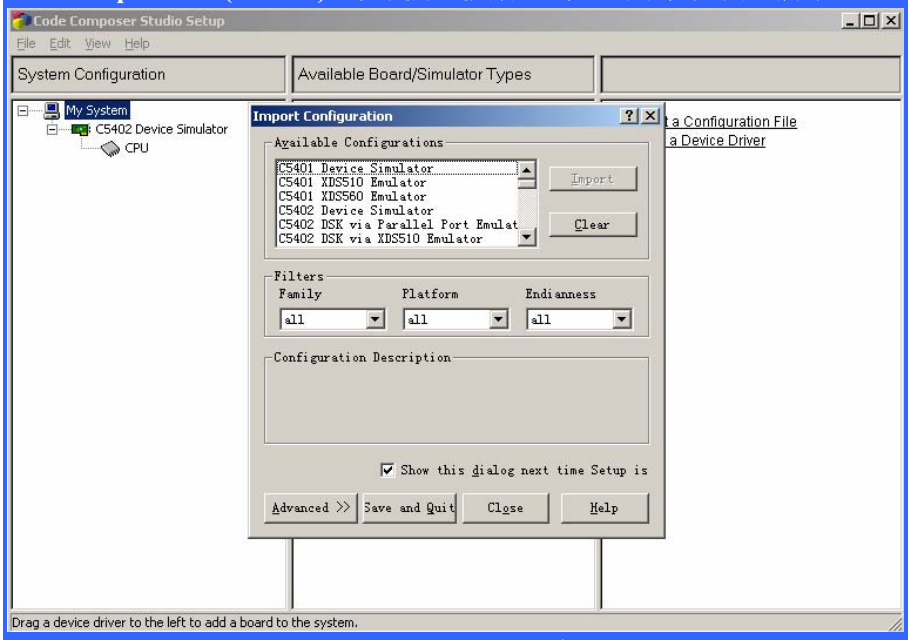

图 1 CCS 配置对话框

用户从“Available Configurations”列表中选取应用平台类型，例如需要使用 C5402 软件仿真器，则选择“C5402 Device Simulator”，然后单击“Import”按钮。对话框中的“Filters”用于设置 DSP 类型、平台类型等。在配置对话框设置完成后，“CCS setup”将“C5402 Device Simulator”作为系统配置显示在“My System”一栏中。

2 CCS 集成开发环境应用

2.1 概述

利用 CCS 集成开发环境，用户可以在一个开发环境下完成工程定义、程序编辑、编译链接、调试和数据分析等工作。使用 CCS 开发应用程序的一般步骤为：

(1) 打开或创建一个工程文件。工程文件中包括源程序（C 或汇编）、目标文件、库文件、连接命令文件和包含文件。关于工程文件的使用请参见 2.3 节。

(2) 使用 CCS 集成编辑环境，编辑各类文件。如头文件（.h 文件），命令文件（.cmd 文件），和源程序(.C，.asm 文件)等。CCS 集成编辑环境使用请参见 2.4 节。

(3) 对工程进行编译。如果有语法错误，将在构建（Build）窗口中显示出来。用户可以根据显示的信息定位错误位置，更改错误。有关编译和调试的进一步描述参见 2.5 至 2.9 节。

(4) 排除程序的语法错误后，用户可以对计算结果/输出数据进行分析，评估算法性能。CCS 提供了探针、图形显示、性能测试等工具来分析数据、评估性能。详细描述参见 2.10、2.11。

2.2 CCS 的窗口、主菜单和工具条

2.2.1 CCS 应用窗口

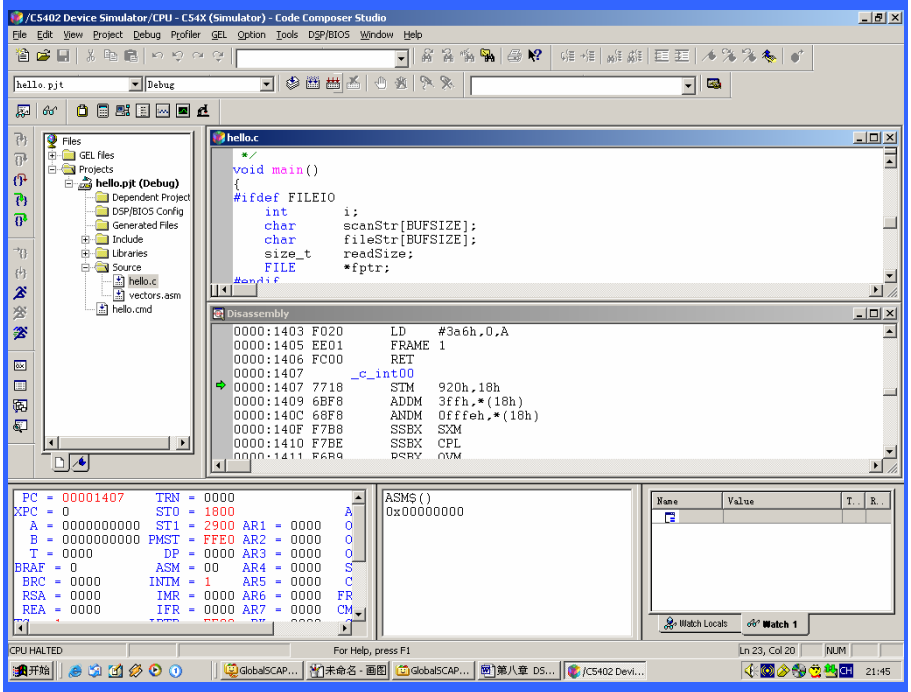

图 2 CCS 应用窗口示例

图 2 为一个典型 CCS 集成开发环境窗口示例。整个窗口由主菜单、工具条、工程窗口、编辑窗口、反汇编窗口、寄存器显示窗口和变量查看

窗口等构成。

工程窗口用来组织用户的若干程序构成一个项目，用户可以从工程列表中选中需要编辑和调试的特定程序。在源程序编辑/调试窗口中用户既可以编辑程序，也可以设置断点、探针，调试程序。反汇编窗口可以帮助用户查看机器指令，查找错误。内存和寄存器显示窗口可以查看、编辑内存单元和寄存器。用户可以通过主菜单 Windows 条目来管理各窗口。

2.2.2 关联菜单

在任一 CCS 活动窗口中单击鼠标右键都可以弹出与此窗口内容相关的菜单。我们称其为关联菜单(Context Menu)。利用此菜单，用户可以对本窗口内容进行特定操作。例如，在 Project View Windows 窗口中单击鼠标右键，弹出图 3 所示菜单。选择不同的条目，用户完成添加程序到工程，打开文件进行编辑，关闭当前工程等功能。

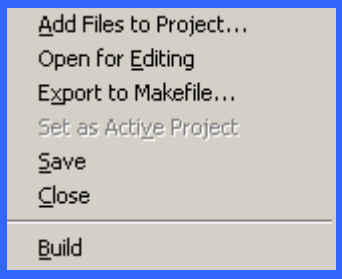

图 3 工程窗口关联菜单

2.2.3 主菜单

主菜单中各选项的使用在后续的章节中会结合具体使用详细介绍，在此仅对菜单项功能作简要说明。用户如果需要了解更详细的信息，请参阅 CCS 在线帮助“Help”。CCS 主菜单如图 4 所示，各项功能解释如表 2 所述。

图 4 CCS 主菜拳

表 2 主菜单简要介绍

菜 单 项	完 成 功 能
File 文件	文件管理，载入执行程序、符号及数据，文件输入/输出等
Edit 编辑	文字及变量编辑。如剪贴操作、字符串查找替换、内存变量、寄存器编辑等
View 查看	工具条显示设置，内存、寄存器和图形显示等
Project 工程	工程管理（新建、打开、关闭及添加文件等）及编译、构建工程等
Debug 调试	断点、探针设置，单步执行、复位等

Profiler 性能	性能菜单。包括时钟和性能设置等
GEL 扩展功能	利用通用扩展语言所设扩展功能菜单
Option 选项	选项设置。设置字体、颜色、键盘属性，动画速度等
Tools 工具	包括管脚连接、端口连接、命令窗口、链接配置等
Window 窗口	窗口管理，包括窗口排列，窗口列表等
Help 帮助	CCS 在线帮助菜单

2.2.4 常用工具条

CCS 将主菜单中常用的命令筛选出来，形成四类工具条：标准工具条，编辑工具条，工程工具条和调试工具条，依次如图 5 至图 8 所示。用户可以单击工具条上的按钮执行相应的操作。

图 5 标准工具条

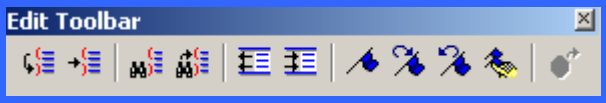

图 6 编辑工具条

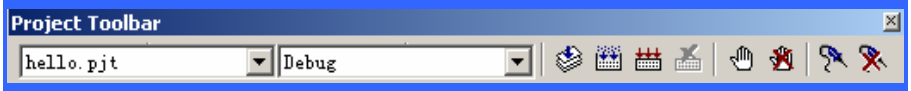

图 7 工程工具条

图 8 调试工具条

标准工具条上的按钮的作用与 Windows 常用按钮相同，在此不做赘述。其他工具条上的按钮作用如表 3 所示。更具体的信息请参阅在线帮助“Help”。

表 3 工具条按钮功能一览

编译工具条	工程工具条	调试工具条
在光标所在处查找括号对	编译当前文件	单步进入
查找下一括号对	增量构建工程	单步执行
查找匹配分支或括号	构建整个工程	单步跳出
查找并定位下一左括号	停止构建工程	执行到光标处
标记的行左突出	设置断点	执行程序
标记的行右缩进	取消所有断点	停止执行

设置或取消标签	设置探针	动画执行
到下一标签	取消所有探针	寄存器窗口
到前一标签		显示内存数据
编辑标签属性		观察堆栈
		反汇编内存

2.3 建立工程文件

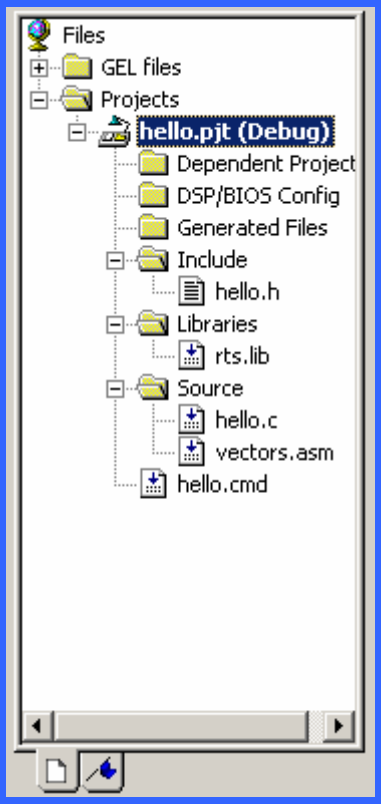

图 9 工程窗口示例

下面按照 CCS 开发应用程序的一般步骤(见 2.1 节所述)，先介绍工程文件的建立与使用。与 Visual Basic、Visual C 和 Delphi 等集成开发工具类似，CCS 采用工程文件来集中管理一个工程。一个工程包括源程序、库文件、链接命令文件和头文件等，它们按照目录树的结构组织在工程文件中。工程构建（编译链接）完成后生成可执行文件。

一个典型的工程文件记录下述信息：

- 源程序文件名和目标库；
- 编译器，汇编器和链接器选项；
- 头文件。

工程视窗显示了工程的整个内容。例如图 9 显示了工程 hello.pjt 所包

含的内容。其中，include 文件夹包含源文件中以“.include”声明的文件，Libraries 文件夹包含所有的后缀为“.lib”的库文件，Source 文件夹包含所有的后缀为“.c”和“.asm”的源文件。文件夹上的“＋”符号表示该文件夹被折叠，“－”表示该文件夹被展开。

2.3.1 创建、打开和关闭工程

命令 Project→New 用于创建一个新的工程文件（后缀为“.pjt”），此后用户就可以编辑源程序、连接命令文件和头文件等，然后加入到工程中。工程编译链接后产生的可执行程序后缀为“.out”。

命令 Project→Open 用于打开一个已存在的工程文件。例如，用户打开位于“d:\ti\ tutorial\sim54xx\hello1”目录下的 hello.pjt 工程文件时，工程中包含的各项信息被载入，其工程窗口如前面图 9 所示。

命令 Project→Close 用于关闭当前工程文件。

2.3.2 在工程中添加/删除文件

以下任一操作都可以添加文件到工程中：

(1) 选择命令 Project→Add Files to Project…

(2) 在工程视图中右键单击调出关联菜单，选择 Add Files…

图 9 所示的 hello.cmd、Source 源文件及 Libraries 库文件需要用户指定加入，头文件（Indude 文件）通过扫描相关性自动加入到工程中。

在工程视图中右键单击某文件，从关联菜单中选择“Remove from project”可以从工程中删除此文件。

2.3.3 扫描相关性

如前所述，头文件加入到工程中通过“扫描相关性”完成。另外，在使用增量编译时（参见 2.5 节“构建工程”），CCS 同样要知道哪些文件互相关联。这些都通过“相关性列表”来实现。CCS 的工程中保存了一个相关性列表，它指明每个源程序和哪些包含文件相关。在构建工程时，CCS 使用命令 Project→Show Dependences 或 Project→Scan All File Dependences 创建相关树。在源文件中以“#include”、“.include”和“".copy”指示的文件被自动加入到工程文件中。

2.4 编辑源程序

CCS 集成编辑环境可以编辑任何文本文件，对 C 程序和汇编程序，还可以彩色高亮显示关键字、注释和字符串。CCS 的内嵌编辑器支持下述功能：

(1) 语法高亮显示。关键字、注释、字符串和汇编指令用不同的颜色显示相互区分。

(2) 查找和替换。可以在一个文件和一组文件中查找替换字符串。

(3) 针对内容的帮助。在源程序内，可以调用针对高亮显示字的帮助。这在获得汇编指令和 GEL 内建函数帮助特别有用。

(4) 多窗口显示。可以打开多个窗口或对同一文件打开多个窗口。

(5) 可以利用标准工具条和编辑工具条帮助用户快速使用编辑功能。

(6) 作为 C 语言编辑器，可以判别圆括号或大括弧是否匹配，排除语法错误。

(7) 所有编辑命令都有快捷键对应。

2.4.1 工具条和快捷键

命令 View→Standard Toolbar 和 View→Edit Toolbar 分别调出标准工具条和编辑工具条。工具条上的按钮含义参见 2.2.4。用户可以根据自己的喜好定义快捷键。除编辑命令外，CCS 所有的菜单命令都可以定义快捷键。选择 Option→Customize…命令打开自定义快捷方式对话框，点击 Keyboard 标签，选中需要定义快捷键的命令。如果此命令已经有快捷键，则在 Assigned 框架中有显示，否则为空白。用户可以点击 Add 按钮，敲下组合键（一般为 Ctrl＋某键），则相应按键描述显示在“Press new shortcut”框中。

2.4.2 查找替换文字

除具有与一般编辑器相同的查找、替换功能外，CCS 还提供了一种“在多个文件中查找”功能。这对在多个文件中追踪、修改变量、函数特别有用。

选择命令 Edit→Find in Files，弹出如下对话框如图 10 所示。分别在“Find what”、“In files of”和“In folder”中键入需要查找的字符串、搜寻目标文件类型以及文件所在目录，然后点击“Find”按钮即可。

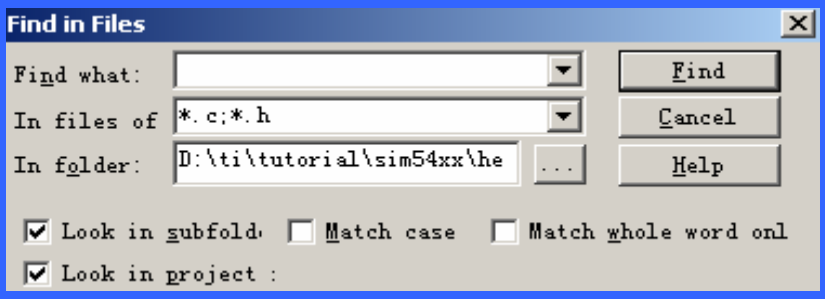

图 10 多个文件中查找字符串

查找的结果显示在输出窗口中，按照文件名、字符串所在行号、匹配文字行依次显示。

2.4.3 使用书签

书签的作用在于帮助用户标记着重点。CCS 允许用户在任意类型文件的任意一行设置书签，书签随 CCS 工作空间（Workspace）保存，在下次载入文件时被重新调入。

1. 设置书签

将光标移到需要设置书签的文字行，在编辑视窗中单击右键，弹出关

联菜单，从“Bookmarks”子菜单中选中“Set a Bookmark”。或者点击编辑工具条的按钮。光标所在行的前端出现一个旗帜，表示标签设置成功。

设置多个书签后，用户可以点击编辑工具条的快速定位书签。

2. 显示和编辑书签列表

以下两种方法都可以显示和编辑书签列表。

(1) 在工程窗口中选择 Bookmark 标签，得到书签列表如图 11 所示。用户可以双击某书签，则在编辑窗口，光标跳转至此书签所在行。右键单击之，用户可以从弹出窗口中编辑或删除此书签。

(2) 选择命令“Edit→Bookmarks”或点击编辑工具条上的按钮。得到图 11 所示书签编辑对话框。双击某书签，则在编辑窗内光标跳转至此书签所在行，同时关闭此对话框。用户也可以单击某书签并且编辑或删除之。

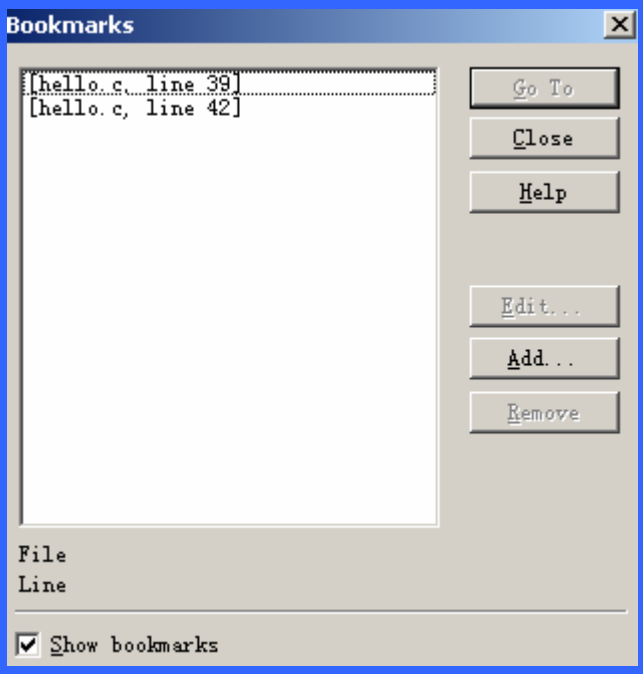

图 11 书签列表

2.5 构建工程

工程所需文件编辑完成后，可以对该工程进行编译链接，产生可执行文件，为调试作准备。CCS 提供了 4 条命令构建工程：

(1) 编译文件：命令 Project→Compile File 或单击工程工具条 按

钮，仅编译当前文件，不进行链接。

(2) 增量构建：单击工程工具条按钮则只编译那些自上次构建后修改过的文件。增量构建（incremental build）只对修改过的源程序进行编译，先前编译过、没有修改的程序不再编译。

(3) 重新构建：命令 Project→Rebuild All 或单击工程工具条按钮重新编译链接当前工程。

(4) 停止构建：命令 Project→Stop Build 或单击工程工具条按钮停止当前的构建进程。

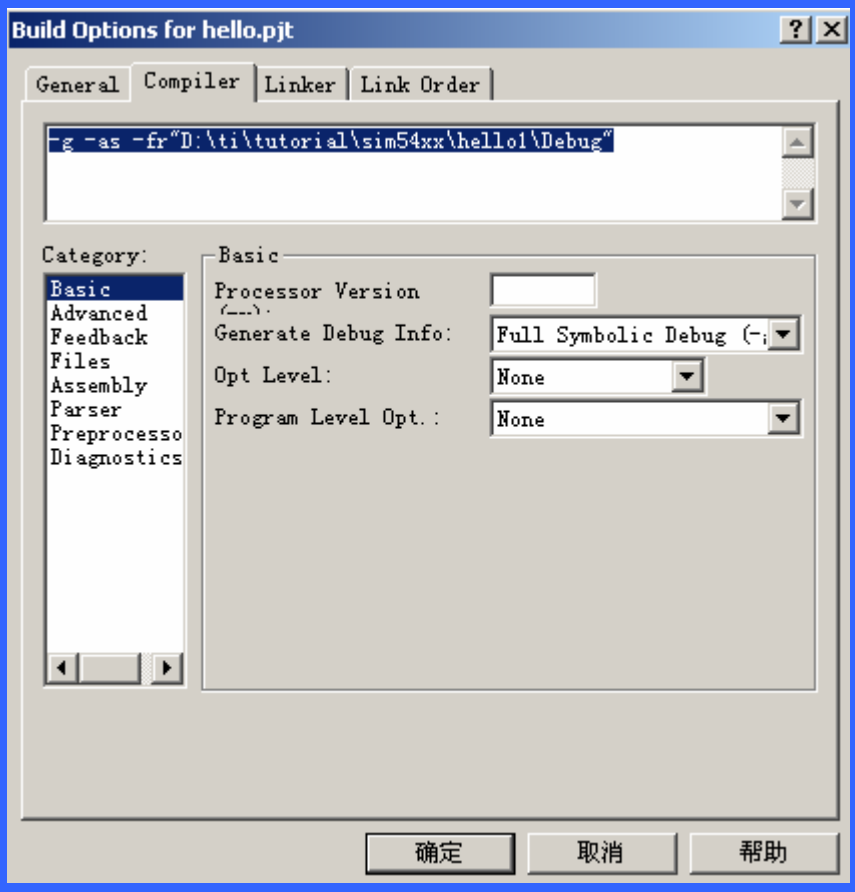

图 12 构建工程选项对话框

CCS 集成开发环境本身并不包含编译器和链接器。而是通过调用第 7 章所述的软件开发工具（C 编译器、汇编器稠链接器）来编译链接用户程序。编译器等所用参数可以通过工程选项设置。选择命令 Project→Build Options 或从工程窗口的关联菜单中选择 Build Options。弹出对话框如图 12 所示。在此对话框中用户可以设置编译器，汇编器和链接器选项，有关

选项的具体含义用户可以参阅第 7 章有关编译器、汇编器和连接器方面的内容，或者查阅联机帮助“Using Code Composer Studio→The Project Environment→Setting Build Options”。

用户也可以对特定的文件设置编译链接选项。操作方法为选择 Project →File Specific Options。然后在对话框中设置相应选项。

2.6 调试

CCS 提供了异常丰富的调试手段。在程序执行控制上，CCS 提供了 4 种单步执行方式。从数据流角度上，用户可以对内存单元和寄存器进行查看和编辑，载入/输出外部数据，设置探针等。一般的调试步骤为：调入构建好的可执行程序，先在感兴趣的程序段设置断点，然后执行程序停留在断点处，查看寄存器的值或内存单元的值，对中间数据进行在线（或输出）分析。反复这个过程直到程序完成预期的功能。

2.6.1 载入可执行程序

命令 File→Load Program 载入编译链接好的可执行程序。用户也可以修改“Program Load”属性，使得在构建工程后自动装入可执行程序。设置方法为选择命令 Options→Customize…，在 Program Load Options 标签下进行设置。

2.6.2 使用反汇编工具

在某些时候（例如调试 C 语言关键代码），用户可能需要深入到汇编指令一-级。此时可以利用 CCS 的反汇编工具。用户的执行程序（不论是 C 程序或是汇编程序）载入到目标板或仿真器时，CCS 调试器自动打开一个反汇编窗口。如图 13 所示。

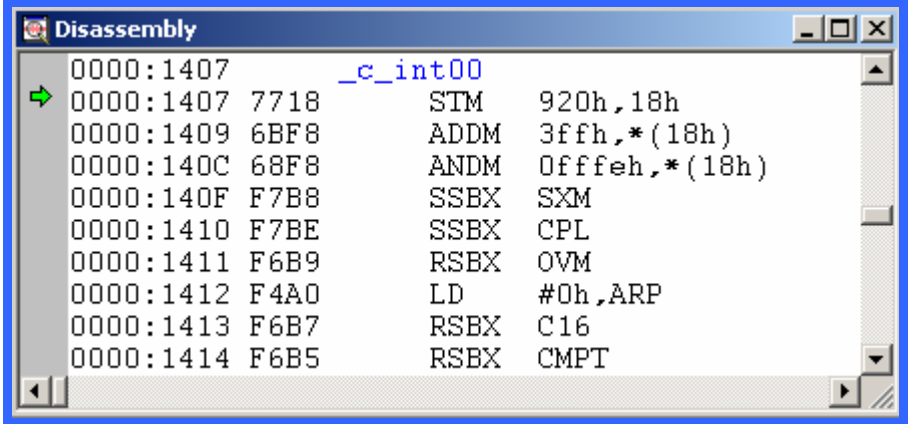

图 13 反汇编窗口

对每一条可反汇编的语句，反汇编窗口显示对应的反汇编指令（某些 C 语句一条可能对应于几条反汇编指令），语句所处地址和操作码（即二进制机器指令）。当前程序指针 PC（Program Pointer）所在语句前有一绿

色箭头指示。当源程序为 C 代码时，用户可以选择使用混合 C 源程序（C 源代码和反汇编指令显示在同一窗口）或汇编代码（只有反汇编指令）模式显示。

除在反汇编窗口中可以显示反汇编代码外，CCS 还允许用户在调试窗口中混合显示 C 和汇编语句。用户可以选择命令 View→Mixed Source/Asm，则在其前面出现一对选中标志。选择 Debug→Go Main，调试器开始执行程序并停留在 main()处。C 源程序显示在编辑窗中，与 C 语句对应的汇编代码以暗色显示在 C 语句下面。

2.6.3 程序执行控制

在调试程序时，用户会经常用到复位、执行、单步执行等命令。我们统称其为程序执行控制。下面我们依次介绍 CCS 的目标板（包括仿真器）复位，执行和单步操作。

l. CCS 提供了 4 种方法复位目标板

(1) Reset DSP：Debug→Reset DSP 命令初始化所有的寄存器内容并暂停运行中的程序。如果目标板不响应命令，并且用户正在使用一基于核的设备驱动，则 DSP 核可能被破坏，用户需要重新装入核代码。对仿真器，CCS 复位所有寄存器到其上电状态。

(2) Load Kernel：如果用户使用一基于核的调试器（不是 JTAG），则 DSP 核应负责与主机通信。若 DSP 核被破坏，则设备驱动程序将无法与目标板通信。在这种情况下，用户应当新装入 DSP 核。命令：Debug→Load Kernel。

(3) Restart：Debug→Restart 命令将 PC 恢复到当前载入程序的入口地址。此命令不执行当前程序。

(4) Go Main：Debug→Go Main 命令在主程序入口处设置一临时断点，然后开始执行。当程序被暂停或遇到一个断点时，临时断点被删除。此命令提供了一快速方法来运行用户应用程序。

2. CCS 提供了 4 种程序执行操作

(1) 执行程序。命令为 Debug→Run 或单击调试工具条上的  按钮。程序运行直到遇到断点为止。

(2) 暂停执行。命令为 Debug→Halt 或单击调试工具条上的  按钮。

(3) 动画执行。命令为 Dehug→Animate 或单击调试工具条上的  按钮。用户可以反复运行执行程序，直到遇到断点为止。

(4) 自由运行。命令为 Debug→Run Free。此命令禁止所有断点，包括探针断点和 Profile 断点，然后运行程序。在自由运行中对目标处理器的任何访问都将恢复断点。若用户在基于 JTAG 设备驱动上使用模拟时，此命令将断开与目标处理器的连接，用户可以拆卸 JTAG 或 MPSD 电缆。

在自由运行状态下用户也可以对目标处理器进行硬件复位。注意在仿真器中 Run Free 无效。

3. CCS 提供的单步执行操作

CCS 提供的单步执行操作有 4 种类型，它们在调试工具条上分别有对应的快捷按钮（参阅 2.2.4）。罗列如下：

(1) 单步进入（快捷键 F8）。命令为 Debug→Step Into 或单击调试工具条上的  按钮。当调试语句不是最基本的汇编指令时，此操作将进入语句内部（如子程序或软件中断）调试。

(2) 单步执行（快捷键 F10）。命令为 Debug→Step Over 或单击调试工具条上的  按钮。此命令将函数或子程序当作一条语句执行，不进其内部调试。

(3) 单步跳出（快捷键 Shift+F7）。命令为 Debug→Step Out 或单击调试工具条上的  按钮。此命令将从子程序中跳出。

(4) 执行到当前光标处（快捷键 Ctrl + F10）。命令为 Debug→Run to Cursor 或单击调试工具条上的  按钮。此命令使程序运行到光标所在的语句处。

2.7 断点设置

断点的作用在于暂停程序的运行，以便观察、修改中间变量或寄存器数值。CCS 提供了两类断点：软件断点和硬件断点。这可以在断点属性中设置。设置断点应当避免以下两种情形：

(1) 将断点设置在属于分支或调用的语句上。

(2) 将断点设置在块重复操作的倒数第一或第二条语句上。

2.7.1 软件断点

只有当断点被设置而且被允许时，断点才能发挥作用。下面我们依次介绍断点的设置，删除断点和断点的使能。

1. 断点设置

有两种方法可以增加一条断点。

(1) 使用断点对话框

选择命令 Debug→Breakpoints 将弹出对话框如图 14 所示。

在“Breakpoint Type”栏中可以选择“无条件断点（Break at Location）”或“有条件断点（Break at Location if expression is TURE）”。在“Location”栏中填写需要中断的指令地址。用户可以观察反汇编窗口，确定指令所处地址。对 C 代码，由于一条 C 语句可能对应若干条汇编指令，难以用唯一地址确定位置。为此用户可以采用“fileName line lineNumber”的形式定位源程序中的一条 C 语句。例如“hello.c line 47”指明在 hello.c 程序第 47 行处语句设置断点。断点类型和位置设置完成后，依次单击“Add”和

“OK”按钮即可。断点设置成功后，该语句前出现红色圆点。

如果用户选择的是带条件断点，则“Expression”栏有效，用户可以按照 2.13 节所述 GEL 语法输入合适的表达式。当此表达式运算结果为真（true=1）时，则程序在此断点位置暂停，否则继续执行下去。

(2) 采用工程工具条

将光标移到需要设置断点的语句上，点击工程工具条上的  按钮。则在该语句位置设置一断点，默认情况下为“无条件断点”。用户也可以使用断点对话框修改断点属性，例如将“无条件断点”改变为“有条件断点”。

2. 断点的删除

在图 14 所示断点对话框中，单击“Breakpoint”列表中的一个断点，然后点击“Delete”按钮即可删除此断点。点击“Delete All”按钮或工程工具条上的  按钮，将删除所有断点。

3. 允许和禁止断点

在图 14 所示断点对话框中，单击“Enable All”或“Disable All”将允许或禁止所有断点。“允许”状态下，断点位置前的复选框有“对勾”符号。注意只有当设置一断点，并使其“允许”时，断点才发挥作用。

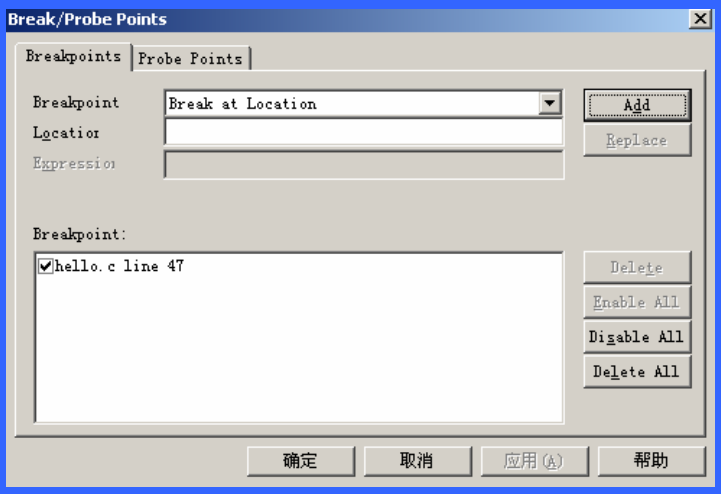

图 14 设置断点对话框

2.7.2 硬件断点

硬件断点与软件断点相比，它并不修改目标程序，因此适用于在 ROM 存储器中设置断点或在内存读写产生中断两种应用。注意在仿真器中不能设置硬件断点。

添加一硬件断点的命令为：Debug→Breakpoint。对两种不同的应用目的，其设置方法为：

(1) 对指令拦截（ROM 程序中设置断点），在断点类型（Breakpoint Type）栏中选择“H/W Breakpoint”。“Location”栏中填入设置语句的地址，其方法与前面所述软件断点地址设置一样。“Count”栏中填入触发计数，即此指令执行多少次后断点才发生作用。依次单击“Add”和“OK”按钮即可。

(2) 对内存读写的中断，在断点类型（Breakpoint Typc）栏中选择“Break on data R/W”。“Location”栏中填入内存地址。“Count”栏中填入触发计数 N。则当读写此内存单元 N 次后。硬件断点发生作用。

硬件断点的允许/禁止和删除方法与软件断点的相同，不再赘述。

2.8 探针断点

CCS 的探针断点提供了一种手段允许用户在特定时刻从外部文件中读入数据或写出数据到外部文件中。2.10 节详细介绍了探针断点的设置与使用，此处略去不述。

2.9 内存、寄存器和变量操作

在调试过程中，用户可能需要不断观察和修改寄存器、内存单元和数据变量。下面，我们依次介绍如何修改内存块，如何查看和编辑内存单元、寄存器和数据变量。

2.9.1 内存块操作

CCS 提供的内存块操作包括拷贝数据块和填充数据块。这在数据块初始化时较为有用。

1. 拷贝数据块

功能：拷贝某段内存到一新位置。

命令：Edit→Memory→Copy，在对活框中填入源数据块首地址、长度和内存空间类型以及目标数据块首地址和内存空间类型即可。

2. 填充数据块

功能：用特定数据填充某段内存。

命令：Edit→Memory→Fill，在对话框中填入内存首地址、长度、填充数据和内存空间类型即可。

2.9.2 查看、编辑内存

CCS 允许显示特定区域的内存单元数据。方法为选择 View→Memory 或单击调试工具条上的  按钮。在弹出对话框中输入内存变量名（或对应地址）、显示方式即可显示指定地址的内存单元。为改变内存窗口显示属性（如数据显示格式，是否对照显示等），可以在内存显示窗口中单击右键，从关联菜单中选择 Properties 即弹出选项对话框。如图 15 所示。

内存窗口选项包括以下内容：

(1) Address：输入需要显示内存区域的起始地址。

(2) Q Value：显示整数时使用的 Q 值（定点位置）。新的整数值＝整

数/2Q。

(3) Format：从下拉菜单中选取数据显示的格式。

(4) Use IEEE Float：是否使用 IEEE 浮点格式。

(5) Page：选择显示的内存空间类型一程序、数据或 I/O。

(6) Enable Reference Buffer：选择此检查框将保存一特定区域的内存快照以便用于比较。例如，用户允许“Enable Reference Buffer”选择，并定义了地址范围为 0x0000～0x002F。此区段的数据将保存到主机内存中。每次用户执行暂停目标板、命中一断点、刷新内存等操作时，编译器都将比较参考缓冲区（Reference Buffer）与当前内存段的内容，数值发生变化的内存单元将用红色突出显示。

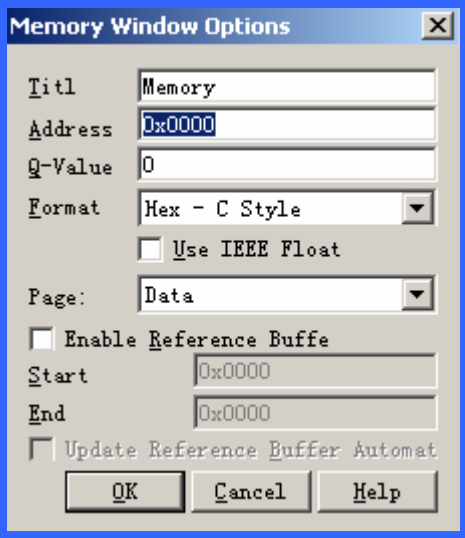

图 15 内存窗口选项对话框

(7) Start Address：用户希望保存到参考缓冲区（Reference Buffer）的内存段的起始地址。只有用户选中“Enable Reference Buffer”检查框时此区域才被激活。

(8) End Address：用户希望保存到参考缓冲区的内存段的终止地址。只有当用户选中“Enable Reference Buffer”检查框时此区域才被激活。

(9) Update Reference Buffer Automatically：若选择此检查框，则参考缓冲区的内容将自动被内存段（由定义参考缓冲区的起始/终止地址所规定的内存区域）的当前内容覆盖。

在“Format”栏下拉条中，用户可以选择多种显示格式显示内存单元，如表 4 所示。

表 4 内存单元数据显示格式

数 据 格 式	描 述
C－style	十六进制字，带前缀“0x”

Hex	TI 格式的十六进制数
Signed integer	有符号整型数
Unsigned integer	无符号整型数
Character	WORD 的低字节作为字符显示
Packed character	每个 word 的高低字节均作为 8Bit 字符显示
Floating piont	十进制浮点显示
Exponential float	指数形式的浮点显示
Binary	二进制显示

编辑某一内存单元的方法为：在内存窗口中鼠标左键双击需要修改的内存单元，或者选择命令 Edit→Memory→Edit，在对话框中指定需要修改的内存单元地址和内存空间类型，并输入新的数据值即可。注意输入数据前面加前缀“0x”为十六进制，否则为十进制。

凡是前面所讲到的需要输入数值（修改地址、数据）的场合，均可以输入 C 表达式。C 表达式由函数名，已定义的变量符号，运算式等构成。下面的例子都是合法的 C 表达式。

例 1　C 表达式举例

MyFunction

0x1000 + 2 * 35

*(mydata+10)

(int) MyFunction + 0x100

PC + 0x10

2.9.3　CPU 寄存器

1. 显示寄存器

选择命令 View→Registers→CPU Register 或单击调试工具条上的  按钮。CCS 将在 CCS 窗口下方弹出一寄存器查看窗口（参见图 2）。

2. 编辑寄存器

有 3 种方法可以修改寄存器的值：

(1) 命令 Edit→Register。

(2) 在寄存器窗口双击需要修改的寄存器。

(3) 在寄存器窗口单击右键，从弹出的菜单中选择需要修改的寄存器。

三种方法都将弹出一编辑对话框，在对话框中指定寄存器（如果在“Register”栏中不是所期望的寄存器）和新的数值即可。

2.9.4　编辑变量

命令 Edit→Variable 可以直接编辑用户定义的数据变量，在对话框中填入变量名（Variable）和新的数值（Value）即可。用户输入变量名后，CCS 会自动在 Value 栏显示原值。注意变量名前应加“*”前缀，否则显

示的是变量地址。在变量名输入栏，用户可以输入 C 表达式（参见例 1），也可以采用“偏移地址@内存页”方式来指定某内存单元。例如：*0x1000@prog，*0x200@io 和*0x1000@data 等。

2.9.5　通过观察窗口查看变量

在程序运行中，用户可能需要不间断地观察某个变量的变化情况。CCS 提供了观察窗口（Watch Window）用于在调试过程中实时地查看和修改变量值。

1. 加入观察变量

选择命令 View→Watch Window，则观察窗口出现在 CCS 的下部位置。CCS 最多提供 4 个观察窗口，在每一个观察窗中用户都可以定义若干个观察变量。有 3 种方法可以定义观察变量：

(1) 将光标移到观察窗口中按“Insert”键，弹出表达式加入对话框，在对话框中填入变量符号即可。

(2) 将光标移到观察窗口中连续两次在 Name 栏单击鼠标（注意不是双击），然后填入变量符号即可。

(3) 在源文件窗口或反汇编窗口双击变量，则该变量反白显示，右键单击选择“Add to Watch Window”。则该变量直接进入当前观察窗口列表。

表达式中的变量符号当作地址还是变量处理取决于目标文件是否包含有符号调试信息。若在编译链接时有 –g 选项（此意味着包含符号调试信息），则变量符号当作真实变量值处理，否则作为地址。对于后一种情况，为显示该内存单元的值，应当在其前面加上前缀星号“*”。

2. 删除某观察变量

有两种方法可以从观察窗口中删去某变量：

(1) 双击观察窗口中某变量，选中后该变量以彩色亮条显示。按“Delete”键，则从列表中删除此变量。

(2) 选中某变量，右键单击，然后选择“Delete Selected Item(s)”。

3. 观察数组或结构变量

某些变量可能包含多个单元，如数组、结构或指针等。这些变量加入到观察窗口中时，会有“＋”或“－”的前缀。“－”表示此变量的组成单元已展开显示，“＋”表示此变量被折叠，组成单元内容不显示。用户可以通过选中变量，然后按回车键来切换这两种状态或直接点击“＋”“－”。

4. 变量显示格式

用户可以在变量名后边跟上格式后缀以显示不同数据格式。例如：MyVar，x 或 MyVar，d 等。允许的数据格式如表 5 所示。

表 5　变量显示格式表

后缀	格　式

d	十进制
e	科学浮点计数法
f	小数浮点数
x	十六进制
o	八进制
u	无符号整数
c	ASCII 字符（字节）
p	大印度格式（Big Endian）打包 ASCII 字符：即第 1 字符在高位字节（MSB byte）上
P	小印度格式（Little Endian）打包 ASCII 字符：即第 1 字符在低位字节（LSB byte）上

用户也可以用“快速观察”按钮来观察某变量。有两种操作方法：

(1) 在调试窗口中双击选中需要观察的变量，使其反白。点击调试工具条上的  按钮。

(2) 选中需要观察的变量后，右键单击从关联菜单中选择“Quick Watch”菜单。

操作完成后，在弹出对话框中单击“Add To Watch”按钮，即可将变量加入到观察窗口变量列表中。

2.10 数据输入与结果分析

在开发应用程序时，常常需要使用外部数据。例如，用户为了验证某个算法的正确性，需要输入原始数据，DSP 程序处理完后，需要对输出结果进行分析。CCS 提供了两种方法来调用和输出数据。

(1) 利用数据读入/写出功能。即调用命令“File→Data（Load/Save）”。这种方式适用于偶尔的、手工读入和写出数据场合。

(2) 利用探针（Probe）功能。即设置探针，通过将探针与外部文件关联起来读入和写出数据。这种方式适用于自动调入和输出数据场合。

2.10.1 载入/保存数据

“载入/保存数据”功能允许用户在程序执行的任何时刻从外部文件中读入数据或保存数据到文件中。需要注意的是，载入数据的变量应当是预先被定义并且有效的。

1. 载入外部数据

程序执行到适当时候。需要向某变量定义的缓冲区载入数据时，选择命令 File→Data→Load 命令，弹出文件载入对话框，选择预先准备好的数据文件（文件的格式应遵循 2.10.3 规定）。此后，弹出一对话框如图 16 所示。“Address”栏和“Length”栏已被文件头信息自动填入。用户也可以在对话栏中重新指定变量名（或缓冲区首地址）和数据块长度。

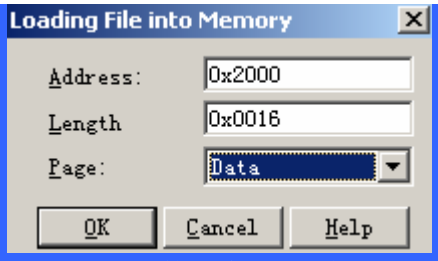

图 16 数据载入对话框

2. 保存数据到文件中

程序执行到适当时候需要保存某缓冲区时，选择命令 File→Data→Save，弹出一对话框要求给出输出文件名。完成后，弹出一“Storing Memory into File”对话框。输入需要保存变量名（或数据块首地址）和长度，单击“OK”按钮即可。

2.10.2 外部文件输入/输出

CCS 提供了一种“探针（probe）”断点来自动读写外部文件。所谓探针是指 CCS 在源程序某条语句上设置的一种断点。每个探针断点都有相应的属性（由用户设置）用来与一个文件的读/写相关联。用户程序运行到探针断点所在语句时，自动读入数据或将计算结果输出到某文件中（依此断点属性而定）。由于文件的读写实际上调用的是操作系统功能，因此不能保证这种数据交换的实时性。有关实时数据交换功能请参考帮助“Help→Tutorial”。

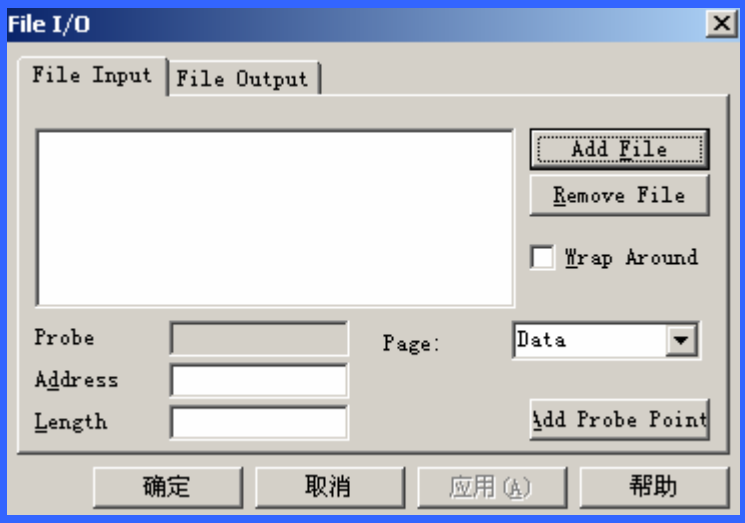

图 17 File I/O 对活框

使用 CCS 文件输入/输出功能遵循以下步骤：

(1) 设置探针断点。将光标移到需要设置探针的语句上，点击工程工具条上的  按钮。光标所在语句前出现一个蓝色的菱形标志。取消设置

的探针，再点击该按钮。此操作仅定义程序执行到何时读入或写出数据。

(2) 选择命令“File→File I/O”，显示对话框如图 17 所示。在此对话中选择文件输入或文件输出功能（对应“File Input”和“File Output”标签）。

假定用户需要读入一批数据。则在“File Input”标签窗口中点击“Add File”按钮，在对话框指定输入的数据文件（假定输入文件为 sine.dat）。数据文件格式遵循 2.10.4 中的规定。注意此时该数据文件并未和探针关联起来，“Probe”栏中显示的是“Not Connected”。

(3) 将探针与输入文件(或者输出文件)关联起来。点击图 17 中的“Add Probe Point”按钮，弹出 Breakpoints/Probe Points 对话框。

• 在“Probe Point”列表中，点击选中需要关联的探针。在本例中只定义了一个探针，故列表中只有一行。

• 从“Connect”一栏中选择刚才加入的数据文件。

• 点击“Replace”按钮。注意在“Probe Point”列表中显示探针所在的行已与文件对应起来。如图 18 所示，

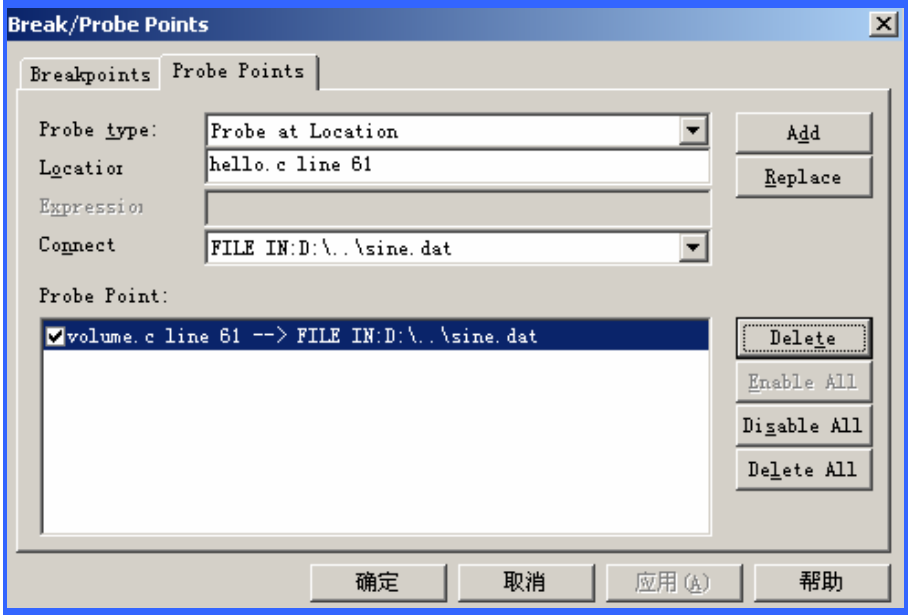

图 18 探针断点设置

(4) Breakpoints/Probe Points 对话框设置完成后，回到图 17 所示“File I/O”对话框。“sine.dat”出现在“File Inport”栏，“Probe”栏中显示的是“Connected”。在此对话框中，指定数据读入存放的起始地址（对文件输出为输出数据块的起始地址）和长度。起始地址可以用事先已定义的缓冲区符号代替。数据的长度以 WORD 为单位。对话框中的“Wrap Around”选项是指当读指针到达文件末尾时，是否回到文件头位置重新读入。这在

用输入数据产生周期信号场合较为有用。

(5) “File I/O”对话框完成后，点击“OK”按钮，CCS 自动检查用户的输入是否正确。

将探针与文件关联后，CCS 给出 File I/O 控制窗口，如图 19 所示。程序执行到探针断点位置调入数据时，其进度会显示在控制窗口内。控制窗口同时给出了若干按钮来控制文件的输入/输出进程。各按钮的作用分别如下所述。

图 19　File I/O 控制窗口

• 运行按钮：在暂停后恢复数据传输；

• 停止按钮：中止所有的数据传输进程；

• 回退按钮：对文件输入，下一采入数据来自文件头位置；对数据输出，新的数据写往文件首部；

• 快进按钮：仿真探针被执行（程序执行探针所在语句）情形。

2.10.3　数据文件格式

CCS 允许的数据文件格式有两种：

(1) COFF 格式，二进制的公共目标文件格式，能够高效地存储大批量数据。

(2) CCS 数据文件。此为字符格式文件，文件由文件头和数据两部分构成。文件头指明文件类型、数据类型、起始地址和长度等信息。其后为数据，每个数据占一行。数据类型可以为十六进制、整数、长整数和浮点数。例 2 给出了一个 CCS 数据文件的头几行内容。

CCS 数据文件文件头格式为：

文件类型	数据类型	起始地址	数据页号	数据长度

解释如下：

文件类型：固定为 1651。

数据类型：取值 1～4，对应类型为十六进制、整数、长整数和浮点数。

起始地址：十六进制，数据存放的内存缓冲区首地址。

数据页号：十六进制，指明数据取自哪个数据页，0 为 PM，1 为 DM，2 为 I/O。

数据长度：十六进制，指明数据块长度，以 WORD 为单位。

例 2　某 CCS 数据文件的头几行内容。

1651　2　20　1　200；数据类型为整数，起始地址 20，存储区为 DM，数据长度为 200。

366

-1479

…

2.10.4 利用图形窗口分析数据

运算结果也可以通过 CCS 提供的图形功能经过一定处理显示出来，CCS 提供的图形显示包括时/频域波形显示、星座图、眼图和图像显示。如表 6 所示。用户准备好需要显示的数据后，选择命令 View→Graph，设置相应的参数，即可按所选图形类型显示数据。

各种图形显示所采用的工作原理基本相同，即采用双缓冲区（采集缓冲区和显示缓冲区）分别存储和显示图形。采集缓冲区存在于实际或仿真目标板，包含用户需要显示的数据区。

显示缓冲区存在于主机内存中，内容为采集缓冲区的拷贝。用户定义好显示参数后，CCS 从采集缓冲区中读取规定长度的数据进行显示。显示缓冲区尺寸可以和采集缓冲区的不同，如果用户允许左移数据显示（Left - Shifted Data Display），则采样数据从显示区的右端向左端循环显示。“左移数据显示”特性对显示串行数据特别有用。

表 6 CCS 图形显示类型

显示类型		描 述
时频图	单曲线图（Single Time）	对数据不加处理，直接画出显示缓冲区数据的幅度－时间曲线
	双曲线图（Dual Time）	在一幅图形上显示两条信号曲线
	FFT 幅度(FFT Magnitude)	对显示缓冲区数据进行 FFT 变换，画出幅度－频率曲线
	复数 FFT(Complex FFT)	对复数数据的实部和虚部分别作 FFT 变换，在一个图形窗口画出两条幅度－频率曲线
	FFT 幅度和相位(FFT Magnitude and Phase)	在一个图形窗口画出幅度—频率曲线和相位－频率曲线
	FFT 多帧显示(FFT Waterfall)	对显示缓冲区数据(实数)进行 FFT 变换，其幅度－频率曲线构成一帧。这些帧按时间顺序构成 FFT 多帧显示图
星座图(Constellation)		显示信号的相位分布
眼图(Eye Diagram)		显示信号码间干扰情况
图像显示(Image)		显示 YUV 或 RGB 图像

CCS 提供的图形显示类型共有 9 种，每种显示所需的设置参数各不相同。限于篇幅，这里仅举例时频图单曲线显示设置方法。其他图形的设置

参数说明请查阅在线帮助“Help→Tutorial”。

选择命令 View→Ghaph→Time/Frequency 弹出 Time/Frequency 对话框，在“Display Type”中选择“Signal Time”（单曲线显示），则弹出图形显示参数设置对话框如图 20 所示。需要设置的参数解释如下：

(1) 显示类型（DisplayType）

单击“DisplayType”栏区域，则出现显示类型下拉菜单条，内容如表 6 所示。点击所需的显示类型，则 Time/Fmquency 对话框(参数设置)相应随之变化。

(2) 视图标题（Graph Title）：定义图形视图标题。

(3) 起始地址（Start Address）

定义采样缓冲区的起始地址。当图形被更新时，采样缓冲区内容亦更新显示缓冲区内容。此对话栏允许输入符号和 C 表达式。当显示类型为“Dual Time”时，需要输入两个采样缓冲区首地址。

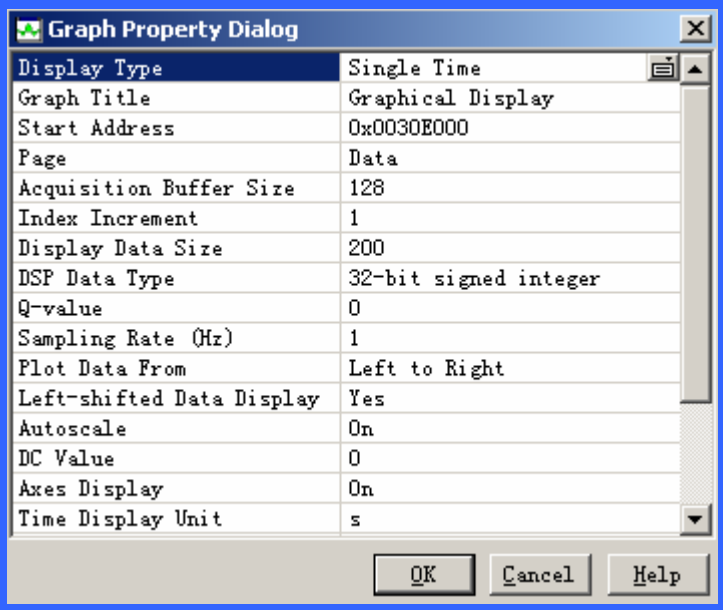

图 20 单曲线显示属性设置参数

(4) 数据页（Data Page）：指明选择的采样缓冲区来自程序、数据还是 I/O 空间。

(5) 采样缓冲区尺寸（Acquisition Buffer Size）

用户可以根据所需定义采样缓冲区的尺寸，例如当一次显示一帧数据时，则缓冲区尺寸为帧的大小。若用户希望观察串行数据，则定义缓冲区尺寸为 1，同时允许左移数据显示。

(6) 索引递增（Index Increment）

定义在显示缓冲区中每隔几个数据取一个采样点。

(7) 显示数据尺寸（Display Data Size）

此参数用来定义显示缓冲区大小。一般地，显示缓冲区的尺寸取决于“显示类型”选项。对时域图形，显示缓冲区尺寸等于要显示的采样点数目，并且大于等于采样缓冲区尺寸。若显示缓冲区尺寸大于采样缓冲区尺寸，则采样数据可以左移到显示缓存显示。对频域图形，显示缓冲区尺寸等于 FFT 帧尺寸，取整为 2 的幂次。

(8) DSP 数据类型（DSP Data Type）

DSP 数据类型可以为：

- 32 比特有符号整数；
- 32 比特无符号整数；
- 32 比特浮点数；
- 32 比特 IEEE 浮点数；
- 16 比特有符号整数；
- 16 比特无符号整数；
- 8 比特有符号整数；
- 8 比特无符号整数。

(9) Q 值（Q-Value）

采样缓冲区中的数始终为 16 进制数，但是它表示的实际数取值范围由 Q 值确定。Q 值为定点数定标值，指明小数点所在的位置。Q 值取值范围为 0～15，假定 Q 值为 xx，则小数点所在的位置为从最低有效位向左数的第 xx 位。

(10) 采样频率（Sampling Rate（Hz））

对时域图形，此参数指明在每个采样时刻定义对同一数据的采样数。假定采样频率为 xx，则一个采样数据对应 xx 个显示缓冲区单元。由于显示缓冲区尺寸固定，因此时间轴取值范围为 0～（显示缓冲区尺寸/采样频率）。

对频域图形，此参数定义频率分析的样点数。频率的取值范围为 0～采样率/2。

(11) 数据给出顺序（Plot Data From）

此参数定义从采样缓冲区取数的顺序：

- 从左至右：采样缓冲区的第一个数被认为是最新或最近到来数据；
- 从右至左：采样缓冲区的第一个数被认为是最旧数据。

(12) 左移数据显示（Left — Shifted Data Display）

此选项确定采样缓冲区与显示缓冲区的哪一边对齐。用户可以选择此特性允许或禁止。若允许，则采样数据从右端填入显示缓冲区。每更新一次图形，则显示缓存数据左移，留出空间填入新的采样数据。注意显示缓冲区初始化为 0。若此特性被禁止，则采样数据简单地覆盖显示缓存。

(13) 自动定标（Autoscale）

此选项允许 Y 轴最大值自动调整。若此选项设置为允许，则视图被显示缓冲区数据最大值归一化显示。若此选项设置为禁止，则对话框中出现一新的设置项“Maximum Y-Value”，设置 Y 轴显示最大值。

(14) 直流量（DC Value）

此参数设置 Y 轴中点的值，即零点对应的数值。对 FFT 幅值显示，此区域不显示。

(15) 坐标显示（Axis Display）

此选项设置 X、Y 坐标轴是否显示。

(16) 时间显示单位（TimeDisplay Unit）

定义时间轴单位。可以为秒（s）、毫秒（ms）、微秒（μs）或采样点。

(17) 状态条显示（Status Bar Display）

此选项设置图形窗口的状态条是否显示。

(18) 幅度显示比例（MagnitudeDisplay Scale）

有两类幅度显示类型：线性或对数显示（公式为 20log（X））。

(19) 数据标绘风格（Data Plot Style）

此选项设置数据如何显示在图形窗口中。

- Line：数据点之间用直线相连；
- Bar：每个数据点用竖直线显示。

(20) 栅格类型（Grid Style）

此选项设置水平或垂直方向底线显示。有 3 个选项：

- No Grid：无栅格；
- Zero Line：仅显示 0 轴；
- Full Grid：显示水平和垂直栅格。

(21) 光标模式（Cursor Mode）

此选项设置光标显示类型。有 3 个选项：

- No Cursor：无光标；
- Data Cursor：在视图状态栏显示数据和光标坐标；
- Zoom Cursor：允许放大显示图形。方法：按住鼠标左键，拖动，则定义的矩形框被放大。图 21 为一余弦波显示图的例子。

2.11 评估代码性能

用户完成一个算法设计和编程后，一般需要测试程序效率以便进一步优化代码。CCS 提供了“代码性能评估”工具来帮助用户评估代码性能。其基本方法为：在适当的语句位置设置断点（软件断点或性能断点），当程序执行通过断点时，有关代码执行的信息被收集并统计。用户通过统计信息评估代码性能。

代码的执行性能通过统计 CPU 执行的指令周期数来完成。假定 CPU

的主频为 f （Hz），平均每执行一条指令需要 n 个周期。经统计某段程序需要花费 M 条指令，并且此段程序必须要在时间 t 秒内完成。

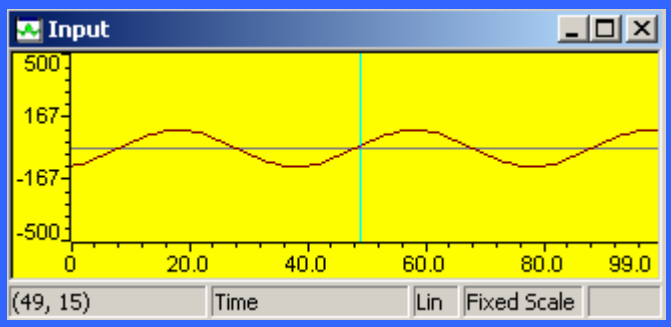

图 21 图形显示举例

则此算法花费时间为：$T = \dfrac{M}{f/n}$ 秒。所需要的 MIPS 为：$P = M/t$。

如果算法花费时间 T 小于限定时间，或者 P 小于 CPU 的 MIPS（f/n），则表明此算法可行。

2.11.1 测量时钟

测量时钟用来统计一段指令的执行时间。指令周期的测量随用户使用的设备驱动不同而变化。假若设备驱动采用 JTAG 扫描通道，则指令周期采用片内分析（on-Chip analysis）计数。

使用测量时钟的步骤为：

(1) 首先允许时钟计数。选择命令 Profile→Enable Clock。有一选中符号出现在菜单项“Enable clock”前面。

(2) 选择命令 Profile→View Clock。则时钟窗口出现在 CCS 主窗口下部位置。

(3) 假定需要测试 A 和 B 两条指令（B 在 A 之后）之间程序段的执行时间。为此，在 B 之后至少隔 4 个指令位置设置断点 C，然后在位置 A 设置断点 A，注意先不要在位置 B 设置断点。

(4) 运行程序到断点 A，双击时钟窗口，使其归零，然后清除 A 断点。

(5) 继续运行程序到 C 断点，然后记录 Clock 的值。其为 A、C 之间程序运行时间 T_1。

(6)用上述方法测量 B、C 断点之间的运行时间 T_2。则（$T_1 - T_2$）即为断点 A、B 之间的执行时间。用这种方法可以排除由于设置断点引入的

时间测量误差。

注意上述方法中设置的是软件断点（有关软件断点的使用见 2.7 节）。

选择命令 Profile→Clock Setup 可以设置时钟属性。弹出对话框如图 22 所示。

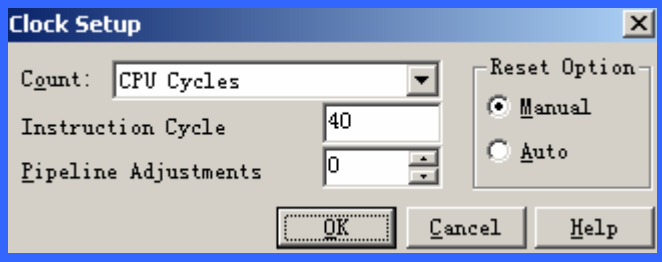

图 22 时钟属性设置

对话框中各输入栏解释如下：

(1) Count：计数的单位。对 simulator，只有 CPU 执行周期（CPU Cycles）选项。

(2) Instruction Cycle：执行一条指令所花费时间，单位为纳秒。此设置将周期数转化为绝对时间。

(3) Pipeline Adjustments：流水线调整花费周期数。当遇到断点或暂停 CPU 执行时，CPU 必须重新刷新流水线，耗费一定周期数。为了获得较好精度的时钟周期计数，需要设置此参数。值得注意的是，CPU 的停止方式不同，其调整流水线的周期数亦不同。此参数设置只能提高一定程度的精度。

(4) Reset Option：用户可以选择手工（Manual）或自动（Auto）选项。此参数设置指令周期计数值是否自动复位（清除为 0）。若选择“自动”则 CLK 在运行目标板之前自动清零，否则其值不断累加。

2.11.2 测试代码执行统计

利用 CCS 的剖析（profile）特性，可以显示程序的有关执行统计。

1. 先加载程序（.out）。

2. 开始一个新的剖析对话。选择命令 Profiler→Start New Session,将弹出 “Profiler Session Name”对话框。

3. 在对话框中输入一个会话名字（默认名为 MySession）。单击 OK 按钮，会出现如图 23 所示的“Profiler View”窗口。

4. 选择需要进行统计的多行指令，单击鼠标右键，在关联菜单中选择 Profile Range→in MySession Session。这样选中的指令就被设置成剖析区域。

5. 执行程序，估计特定代码段执行完后（或者在代码段尾部设置一软件断点），终止运行。则在统计窗口中出现统计的信息。统计数据的含义如表 7 所示。

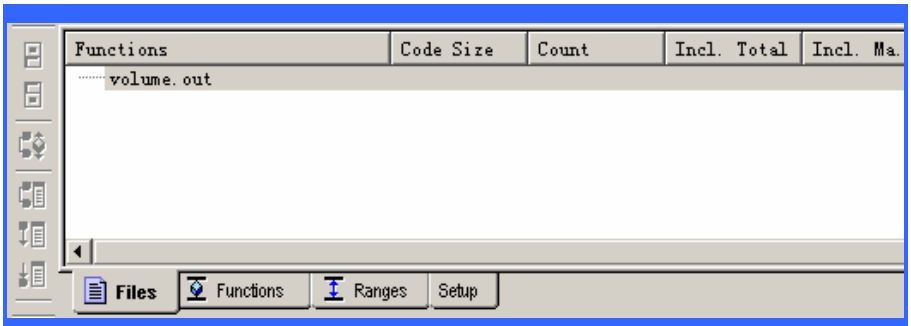

图 23 程序执行信息统计表

表 7 统计信息栏目含义

栏 目	描 述
Code Size	最小二进制指令数
Count	剖析区域在程序运行时进行统计的次数
Incl.Total	总统计值
Incl.Maximum	最大统计值
Incl.Minimum	最小统计值
Excl.*	含义同 Incl.*，只是其中不包括调用子程序的统计值

2.12 内存映射

内存映射规定了用户代码和数据在内存空间的分配。一般地，用户在链接命令文件（.cmd）中定义内存映射表，除此之外，CCS 还提供了在线手段来定义内存映射。用户允许“内存映射”时，CCS 调试器检查每一条内存读写命令，看它是否与定义的内存映射属性相矛盾。若用户试图访问未定义内存或受保护区域，则 CCS 仅显示其默认值，而不访问内存。

2.12.1 查看和定义内存映射

选择命令 Option→Memory Map，弹出对话框如图 24 所示。用户可以利用对话框查看和定义内存映射。在默认情况下，“Enable Memory Mapping”复选框是未选中的，目标板上所有 RAM 都是有效可访问的。为利用内存映射机制，确保“Enable Memory Mapping”复选框选中(单击复选框，前面出现√符号)。选择需要定义的内存空间（代码、数据或 I/O）。在“Starting Address”和“Length”栏中输入需要映射的内存块起始地址和长度，选择读/写属性。单击“Add”按钮，则新的内存映射定义被输入。用户也可以选中一个已定义好的内存映射并删除之。

用户新定义的内存区域可以和以前定义的相重叠，重叠部分的属性按新定义的来算。注意“Reset”按钮将禁止所有内存单元的读写。

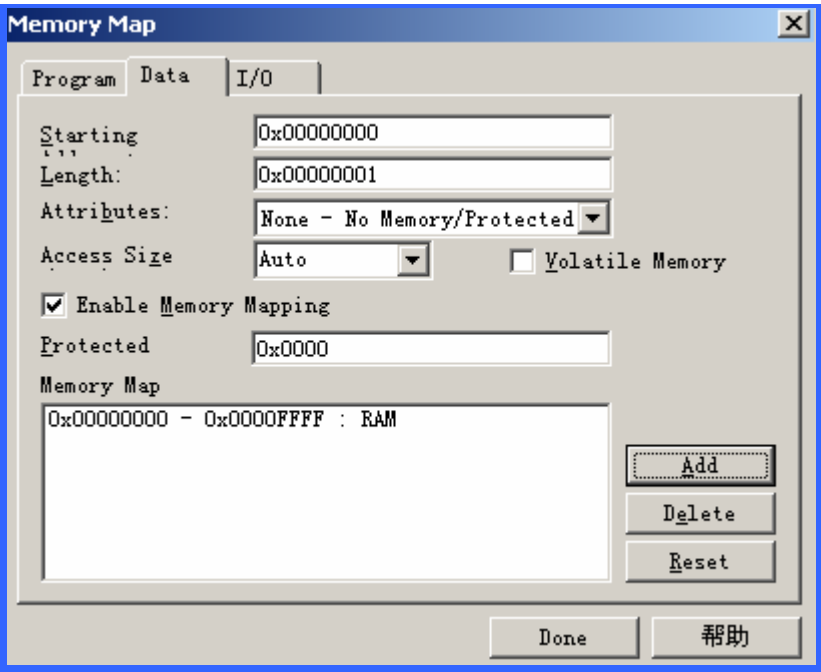

图 24 内存映射对话框

2.12.2 利用 GEL 来定义内存映射

用户可以利用 GEL 文件来定义内存映射。在启动 CCS 时，将 GEL 文件名作为一个参数引用 ，则 CCS 自动调入 GEL 文件，允许内存映射机制。

下列 GEL 函数可用来定义内存映射，如表 8 所示。

表 8 用来定义内存映射的 GEL 函数列表

函数	解释	语 法
GEL-MapAdd( )	增加内存映射	GEL-MapAdd( )（地址、内存页、长度、可读、可写）
GEL-MapDelete( )	删除内存映射	GEL-MapDelete( ) （地址、内存页）
GEL-MapOn( )	允许内行映射	GEL-MapOn( )
GEL-MapOff( )	禁止内存映射	GEL-MapOff( )
GEL-MapReset( )	复位内存映射	GEL-MapReset( )

下面给出一个利用 GEL 函数定义内存映射的例子。在本例中，程序空间和数据空间[0:0xF000]内存段被定义为可读可写。

例 3 利用 GEL 函数来定义内存映射。

```
StartUp( )
{
GEL-MapOn( )
```

GEL-MapReset( )

GEL-MapAdd(0，0，0xF000，1，1);

GEL-MapAdd(0，l，0xF000，1，1);

}

2.13 通用扩展语言 GEL

GEL（General Extension Language）通用扩展语言是一种与 C 类似的解释性语言。利用 GEL 语言，用户可以访问实际/仿真目标板，设置 GEL 菜单选项，特别适用于自动测试和自定义工作空间。

2.13.l GEL 函数调用

CCS 提供了 3 种方法调用 GEL 函数：在命令行中，在 GEL 菜单中和自动调用 GEL 函数。

l. 在命令行中调用 GEL 函数

选择命令 Edit→Command Line，弹出命令行对话框，输入调用 GEL 函数的命令后，用户可以在 GEL 工具条中选定命令行（命令 View→GEL Toolbar 使能 GEL 工具条），并执行之。

2. 在 GEL 菜单中调用 GEL 函数

其方法为：先载入包含 GEL 函数的.gel 文件。GEL 函数驻留在 CCS 内，直到用户从工程中删除 GEL 文件。GEL 加载器在载入 GEL 文件时检查 GEL 函数语法，并显示可能的错误信息。只有改正所有错误后，CCS 才能使用 GEL 函数。选择命令 File→Load GEL 或在工程窗口中右键单击“GEL Files”文件夹，从关联菜单中选择“Load GEL”可以载入 GEL 文件。GEL 文件调入后，GEL 函数自动出现在主菜单的 GEL 下拉菜单中。

3. 自动调用 GEL 函数

CCS 提供了一个名为 StartUp 的 GEL 函数，可以在 CCS 启动时自动载入运行。利用此函数，用户可以建立起所需的工作环境。操作步骤为：

(1) 建立 GEL 文件，例如 myfile.gel。其中包含有 StartUp( )函数。

(2) 修改 CCS 快捷方式图标属性，将 myfile.gel 作为 CCS 的命令参数。例如修改为 d:\ti \c\cc\bin\cc-app.exe myfile.gel。

此后用户单击 CCS 快捷方式图标，则 myfile.gel 里的 StartUp 函数被检索并执行。前面所述的例 3 即为一个 StartUp GEL 函数例子。

2.13.2 GEL 语法

GEL 是 C 编程语言的子集。在 GEL 程序中，用户不能声明参数变量，所有变量都是由 DSP 程序定义并存在于实际仿真目标板中。GEL 函数就是利用这些已有变量完成相应的功能。

GEL 函数定义形式为：

FuncName([parameter1[,parameter2…[,parameter6]]])

{statements }

其中参数说明如下：

• funcName:GEL 函数名称；

• parameters:GEL 函数参数；

• statements:GEL 函数语句。

在下例 GEL 函数调用中，一个 DSP 符号变量被赋值，并调入一文件，返回一常数值。调用方式 Initialize(targetSymbol，“c:\\myfile.out”，23*5+1.22)。

例 4　GEL 函数参数使用。

```
Initialize(a,filename,b)
{ targVar=0;
a=0;
Gel_Load(filename);
Return b*b; }
```

Initialize GEL 函数执行后，“targetSymbol”变量被赋值为 0，“myfile.out”被调入 CCS 中，参数“b”赋予常数 116.22。在执行 GEL 函数之前，必须保证参数列表中的 DSP 符号信息已经被载入到 CCS 中。例如在例 3 中执行 Initialize 函数之前，包含“targetSymbol”符号信息的 COFF 文件应已经被载入到 CCS 中。GEL 函数的语句类型如表 9 所示。

表 9　GEL 函数语句类型

GEL 函数语句	语法	应用说明
返回语句	return expression	返回数据

If-Else 语句	if(expression)
	statement1
	else
	statement2
		若表达式(expression)为真，则执行语句 l。否则，执行语句 2

While 语句	while(expression) statement	若表达式(expression)为真，则执行语句
注释	/*注释语句*/	注释行
预处理语句	# define identifier token - sequence	类似于 C 预处理语句

2.14 工具的使用

下面介绍一下 Tools 菜单下的三个常用工具的使用：

2.14.l　端口链接 Port Connect

Port Connect 工具可以通过 I/O 端口地址来访问一个文件。通过将一个 I/O 端口地址和一个文件关联，就可以从该文件中读入数据，也可以把

数据输出到该文件中。

具体操作步骤如下：

首先在当前的 gel 文件中定义地址映射：

GEL_MapAdd(address, page, length, readable, writeable);

其中，address 为地址值；Page 为存储空间值（Memory Type Value），0 为程序存储空间（Program memory ），1 为数据存储空间（Data memory），2 为 I/O 空间（I/O space）；length 为地址空间长度，对于 I/O 空间一般取为 1；readable 为可读标志，0 为不可读，1 为可读；writeable 为可写标志，0 为不可写，1 为可写。

接着做如下操作：

1）在 Tools 菜单下选择 Port Connect；

2）在出现的窗口中单击 Connect 按钮将弹出 Connect 对话框；

3）在 Port 栏内填入内存地址，可以是绝对地址、C 表达式、C 函数名、汇编语言标号。（注意：十六进制地址前必须加上前缀 0x，否则，CCS 将把它当作十进制地址来对待）；

4）在 Length 栏内填入端口范围的大小，该长度可以是任意的 C 表达式，一般为 1。

5）在 Page 下拉栏内选择地址覆盖的内存类型：Prog（程序存储区）、Data（数据存储区）、I/O（I/O 空间）。

6）在 Type 栏内点击只读或只写单选按钮。

7）单击 OK 按钮将弹出 Open Port File 窗口。

8）选择你想要和端口连接的数据文件并单击 Open。

这样，在用汇编语言读或写相应的 I/O 端口地址时就可以访问输入或输出数据文件了。

当然，也可以在当前的 gel 文件中定义如下 I/O 链接，来替代上述 8 个步骤用：

GEL_PortConnect(portAddress, page, length, accessType, "fileName");

其中，portAddress 为 I/O 地址值；Page 和 length 同上；accessType 为存取类型，0x01 为端口读，0x02 为端口写。

例 如 ： GEL_PortConnect(0x0001, 2, 0x001,1 , "c:\\mydir\\myfile.dat");

该 I/O 链接是实现从 I/O 端口 1 读入“myfile.dat”文件中的数据，而不再需要使用 Port Connect 命令，就会自动添加 Port connect 关联。不过如果要重新从头读入数据，就要重新加载当前的 gel 文件。

2.14.2 引脚链接 Pin Connect

仿真器允许你仿真和监视外部的中断信号，Pin Connect 工具就是用

来让你设置发生的中断信号的时间间隔。

仿真外部中断的步骤如下：

1） 创建一个数据文件用来指定中断间隔时间。

为了仿真外部中断，你必须首先创建一个数据文件用来指定中断间隔时间。该中断间隔时间表示的是 CPU 时钟周期数，可以是绝对时钟周期，也可以是相对时钟周期（相对于上一个事件发生的时钟周期数）。在第一个时钟周期到来时将开始仿真，然后在每个指定的时钟周期将产生一个中断。

2） 将一个外部中断管脚和该数据文件关联起来。

选择 Tools 菜单下的 Pin Connect 命令后将出现一个窗口，可用的外部中断管脚将列在 Pin Name 栏下，注意到此时的文件名栏下显示的是“Not Connected”。单击你想要使用的管脚名后再单击 Connect 按钮（或直接双击管脚名）将出现一个 Open Pin File 对话框，选择定义了中断间隔时间的数据文件即可。这样就将数据文件和选定的管脚关联在一起了。这时 Filename 栏下出现了刚刚选择的数据文件。

3） 加载程序；

4） 运行程序。

2.14.3 链接器设置 Linker Configuration

用来选择工程所需的链接命令文件。

步骤如下：

1） 选择 Tools 菜单下的 Linker Configuration。

2） 在弹出的对话框中选择你所需的链接命令文件的类型（可视化链接命令文件或文本链接命令文件）。

3） 点击 OK。

4） 选择 Project→Add Files to Project，在弹出的对话框中找寻你所需的链接命令文件（可视化链接命令文件扩展名为.rcp，文本链接命令文件扩展名为.cmd）。

3 CCS 使用举例

在第 2 节中介绍了 CCS 常用工具和命令，下面结合一个具体的例子来介绍 CCS 的使用。

利用 CCS 开发应用软件的大致流程如图 25 所示。

各阶段完成功能如下：

(1) 软件设计：程序模块划分，算法和流程确定，预测执行结果等。

(2) 程序编辑和编译：创建工程文件，编写头文件、配置文件和源程序，使用汇编或 C 编译器进行编译，排除语法、变量定义等错误。

(3) 程序调试：利用单步执行。断点、探针等手段调试程序。

(4) 输出结果分析：利用 CCS 提供的工具分析 DSP 程序的运行结果，如图形显示数据或统计运行时间等。若算法不能满足要求，则重新进行软件设计。

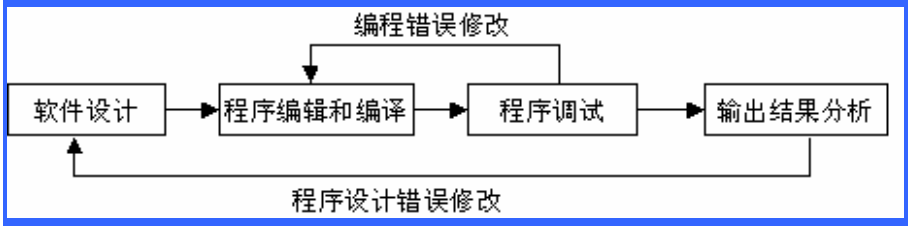

图 25　CCS 软件开发流程

CCS 使用举例：任意信号发生器。

该例子设计产生正弦波、余弦波、方波、三角波的信号。通过设计改变 5402 DSK 实验板上用户选择开关 DIP 中的 S1 的状态，产生不同的信号。

在具体实验开始前，先利用 MATLAB 语言与开发环境或其他高级语言与开发环境进行函数信号发生器的高级语言设计与仿真，然后利用仿真结果得出的数据进行以下的工作。

对于 DSP 的 CCS 开发环境一般支持对 C 语言和汇编语言的编译处理，虽然信号的产生有多种方法，但是在此与一般 DSP 设计相同，利用 C 语言和汇编语言相结合来设计该信号发生器。

首先用 C 语言设计信号的产生部分，由于 C 语言简洁易懂、使用方便灵活，还能直接访问物理地址，进行位操作，能实现汇编的大部分功能，所以也可用于对硬件进行操作，而且 C 语言可移植性好，,基本上不作修改就能用于各种型号的计算机和操作系统。鉴于此，在 CCS 环境下能用 C 语言直接进行设计和仿真。在 C 语言中设计好了正弦波、余弦波、方波、三角波函数，再加入一外部调用函数 select()，以便调用汇编语言产生的信号控制部分。

其次，由 select()调用汇编程序进行信号产生的控制，汇编语言可以有效的与硬件相结合，对硬件直接操作，可大大地提高其运行速度。信号采样进入预先定义的数组中，由于 DIP 中有两个按钮的状态值可由用户自定义，两个按钮可产生四种不同的状态，通过对 DIP 开关的控制即可产生输出上述四种信号。运行时通过硬件板读入 DIP 值，根据 DIP 状态决定显示信号类型，将存储该信号采样点的数组中的值读入一个专门放信号的数据存储器中预先定义的一个数据段。

????

该例子中的 CCS 运行界面如图 26 所示。

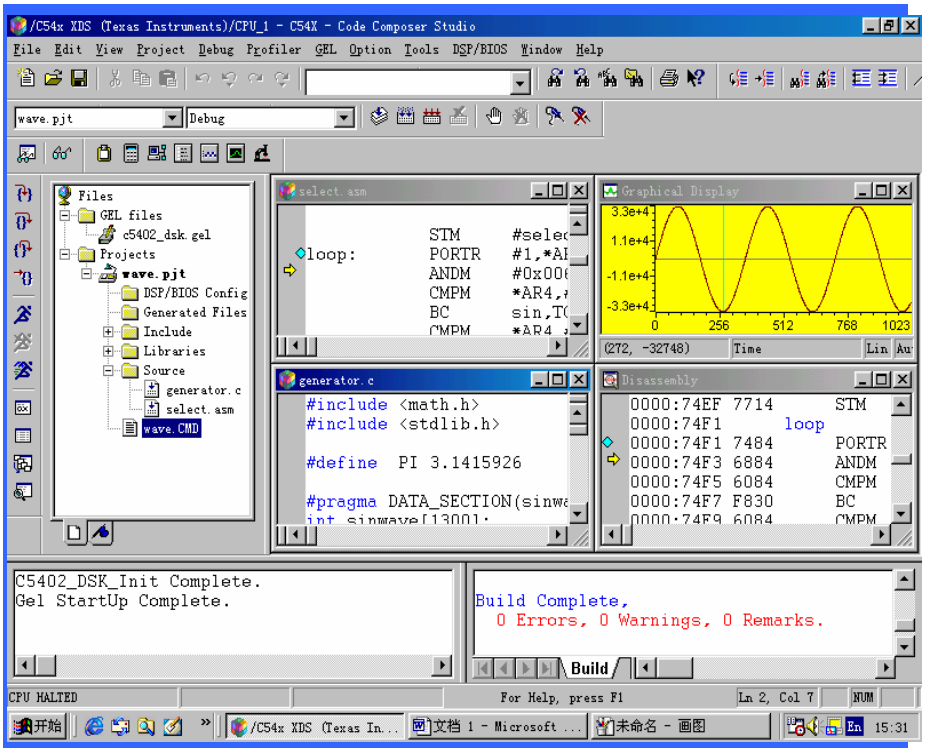

图 26　该例子中的 CCS 界面

该例子的输出图形：

(1) 输出为正弦信号(见图 27，幅度周期的设定在 C 语言程序中)。

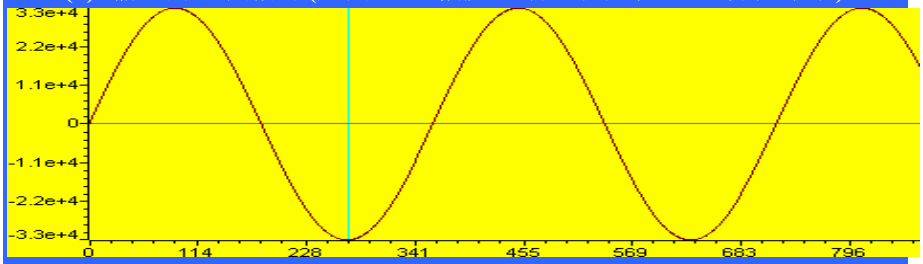

图 27　正弦信号

(2)输出为余弦信号(见图 28)。

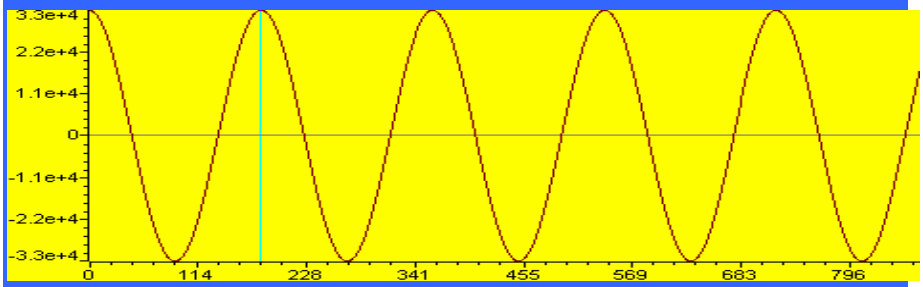

图 28　余弦信号

(3) 输出为方波信号(见图 29)。

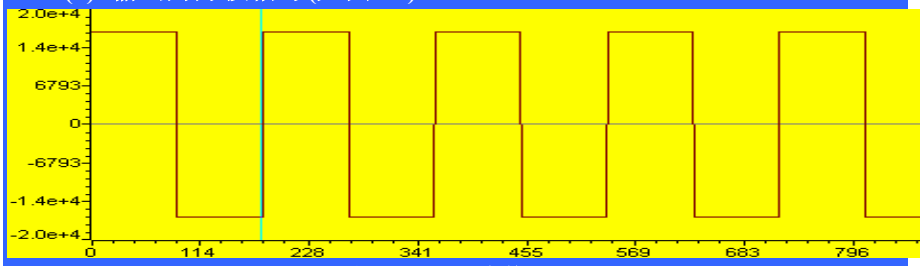

图 29　方波信号

(4) 输出为三角波信号(见图 30)。

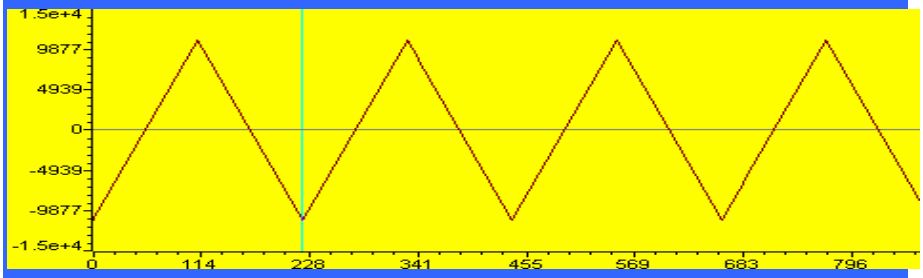

图 30　三角波信号

主要程序代码如下：

例5　任意信号发生器主程序：

```
                .title      "codecinit.asm"
                .mmregs
                .include    "selfdefine.h"
                .ref        _select
                .def        _c_int00
                .def        _codecinit
                .def        codecreceiveint
                .ref        outbuffer
                .ref        selection
_c_int00:
codecreceiveint:    LDM     DRR11, A
                    LD      *AR2+, B
                    AND     #0XFFFE, B
                    STLM    B, DXR11
                    BANZ    NEXT2, *AR1-
                    STM     #outbuffer, AR2
                    STM     #1024, AR1
NEXT2:              NOP
```

```
        RETE
_codecinit:
        STM     #0000H, DMPEC
        SSBX    INTM
        LD      #0, DP
        STM     #0a0H, PMST
        STM     #0800H, IMR
        STM     #0FFFFH, IFR
        STM     #SPCR11_SUBADD, SPSA1
        STM     #0000h, SPSD1
        STM     #SPCR21_SUBADD, SPSA1
        STM     #0000h, SPSD1                   ;复位串行缓冲口
        STM     #RCR11_SUBADD, SPSA1        ;RCR
        STM     #0040h, SPSD1
        STM     #RCR21_SUBADD, SPSA1
        STM     #0000h, SPSD1
        STM     #XCR11_SUBADD, SPSA1        ;XCR
        STM     #0040h, SPSD1
        STM     #XCR21_SUBADD, SPSA1
        STM     #0000h, SPSD1
        STM     #SRGR11_SUBADD, SPSA1       ;SRGR
        STM     #0000h, SPSD1
        STM     #SRGR21_SUBADD, SPSA1
        STM     #0000h, SPSD1
        STM     #PCR1_SUBADD, SPSA1         ;PCR
        STM     #000Ch, SPSD1
        STM     #selection, AR7
        ST      #0x40,*AR7   ;0 sin 20 cos 40 square 60 tri
        CALL    _select
        STM     #outbuffer, AR2
        STM     #1024, AR1
        RSBX    INTM
        STM     #SPCR11_SUBADD, SPSA1
                                    ;使采样率发生器退出复位状态
        STM     #0001h, SPSD1
        STM     #SPCR21_SUBADD, SPSA1
        STM     #0001h, SPSD1
```

```
EDNL:       NOP
            B    EDNL
            RET
任意信号发生器中信号选择子程序：
            .title      "select.asm"
            .mmregs
            .global
      _sinwave,_coswave,_squarewave,_trianglewave
            .global   _select
            .def      outbuffer
            .def      selection
STACK:      .usect    "STACK", 20
outbuffer:        .usect    "outbuffer", 1024
selection:  .usect    "selection", 1
            .text
_select:    LD        #0,   A                    ;clear buffer
            STM       #outbuffer, AR2
            RPT             #1023
            STL       A, *AR2+
            STM       #selection, AR4
loop:       ANDM      #0x0060, *AR4
            CMPM      *AR4, #0000
            BC        sin, TC
            CMPM      *AR4, #0x0020
            BC        cos, TC
            CMPM      *AR4, #0x0040
            BC        square, TC
            CMPM      *AR4, #0x0060
            BC        triangle, TC
sin:        STM       #outbuffer, AR5
            STM       #_sinwave, AR3
            RPT       #1023
            MVDD      *AR3+, *AR5+
            B         out
cos:        STM       #outbuffer, AR5
            STM       #_coswave, AR3
            RPT             #1023
```

```
        MVDD   *AR3+, *AR5+
        B      out
square: STM    #outbuffer, AR5
        STM    #_squarewave, AR3
        RPT        #1023
        MVDD   *AR3+, *AR5+
        B      out
triangle: STM  #outbuffer, AR5
        STM    #_trianglewave, AR3
        RPT        #1023
        MVDD   *AR3+, *AR5+
        B      out
out:    NOP
        RET
```

2. 构造工程及输入源程序

启动 CCS，选择命令 Project→New。创建一个新的工程 signal_generator.pjt。点击标准工具条上的  按钮，编辑输入命令链接文件 signal_generator.cmd，包含源文件 select.asm、codecinit.asm、generator.c 和 vectors.asm。文件编辑完成后，将上述文件添加到 signal_generator 工程中。加入方法参见 2.3.2。

3. 编译和调试

选择命令 Project→Build 命令，编译链接 signal_generator 项目。若工程构建成功。则在当前翻目录下生成可执行文件对 signal_generator.out。选择命令 File→Load Program，载入 signal_generator.out，反汇编窗口自动打开。

调试分为两个阶段。第一阶段先验证输出信号的第一个采样点是否正确。第二阶段检查 200 个输出点是否正确。

(1) 第一阶段的验证分为三个步骤。首先查看计算出的第一个载波采样点 sin(π/8)是否正确，再查看计算调幅信号的第一个采样点 sin（π/160）是否正确结果，最后查看二者的乘积是否正确。为此在主程序的三个位置设置断点（断点设置参见 2.7.1），如图 31 所示。

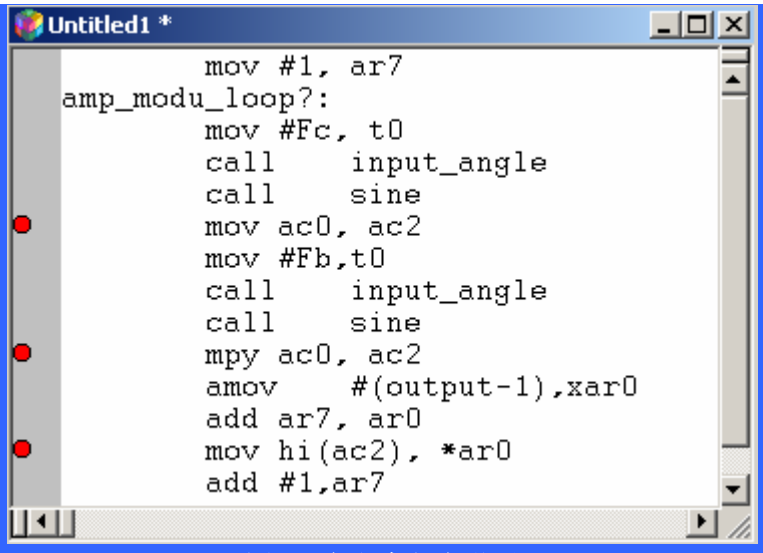

图 31 主程序断点设置

sin 函数的输出结果存放在 AC0 寄存器高 16 位中，由于调用两次 sin 函数，因此将其值分别存入 AC2 和 AC0 中。程序执行到断点 2 时，选择命令 View→Registers→CPU Registers 显示寄存器查看窗口。DSP 计算出的采样数据分别为 sin(π/8)＝0x2FE8(AC2 高 16 位，Q15)＝0.3743 和 sin(π/160)=0x0278(AC0 高 16 位，Q15)=0.0193。标准值为 sin(π/8)=0.3827，sin(π/160)=0.0196。二者基本一致，误差是由相角的定点而引起的。

(2) 在主程序计算完成后，设置一断点（例如在程序尾部的“nop”语句上设置断点）。观察计算出的 200 个采样点数据是否正确。

4. 输出结果分析

为直观起见，先根据计算结果观察信号波形。选择命令 View→Graph→Time/Frequency，定义显示图形类型为“Single Time”。在“Start Address”、“Acquisition Buffer Size”和“Display Data Size”栏目分别填入“output”（显示数据起始地址）、“200”（采样数据大小）、“200”（显示数据大小）。数据类型定义为 16 比特有符号整数和 Q15 格式。其他参数使用默认值。更具体的图形显示参数设置描述参见 2.10.4。图形显示其形状为一正弦调幅信号。将 200 个点的计算结果与标准值相对照，其误差最大值为 0.002，平均误差为 0.0001，满足精度要求。

5. 程序性能分析

为考察 sin 函数的执行效率，用户可以使用 2.11 节所述“代码性能评估工具”来统计 sin 函数的执行周期数。选择第一个 sine 函数语句作为剖析区域，同时允许时钟统计（选择命令 Profiler—>Enable Clock），则程序运行信息统计窗口出现在 CCS 主窗口下部位置。运行程序，得到性能统计信息图。由图中可知，sin 函数平均耗费 DSP CPU 指令周期数为 59。假

定 CPU 主频为 120MHz，平均一个周期执行一条指令，则计算一个正弦函数值需花费 492ns。

本节通过一个典型实例讨论了 CCS 的使用过程。从上述讨论中可知，利用 CCS 可以在一个软件环境下一次性完成编辑、编译、调试和数据分析等工作。充分利用 CCS 所提供的功能，可以帮助我们高效地开发 DSP 应用系统。