第 5 章　DSP 集成开发环境(CCS)

教学提示：Code Composer Studio 简称 CCS，是 TI 公司推出的为开发 TMS320 系列 DSP 软件的集成开发环境（IDE）。CCS 自推出以来发展出了多个版本，本章以 CCS2.0 为参照进行讲述。

教学要求：了解 CCS 的软件开发流程和 CCS 环境具有的功能，能够操作 CCS 的窗口、菜单和工具条。掌握 CCS 工程管理的概念，能够完成简单程序的编辑、汇编、连接和调试，并掌握探针和显示图形的使用。了解用 Simulator 仿真中断和仿真 I/O 端口的方法。

5.1　CCS 集成开发环境简介

CCS 工作在 Windows 操作系统下，类似于 VC++的集成开发环境，采用图形接口界面，有编辑工具和工程管理工具。它将汇编器、链接器、C/C++编译器、建库工具等集成在一个统一的开发平台中。CCS 所集成的代码调试工具具有各种调试功能，能对 TMS320 系列 DSP 进行指令级的仿真和可视化的实时数据分析。此外，还提供了丰富的输入/输出库函数和信号处理的库函数，极大地方便了 TMS320 系列 DSP 软件开发过程。

C5000 CCS 是专门为开发 C5000 系列 DSP 应用设计的，包括 C54x 和 C55x DSP。

利用 CCS 的软件开发流程如图 5.1 所示。

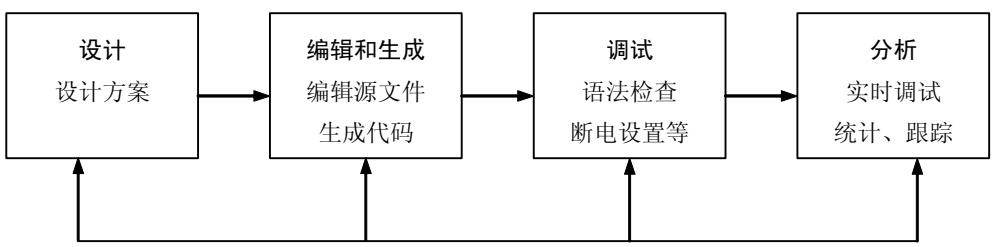

图 5.1　CCS 的软件开发流程

5.1.1　CCS 安装及设置

1. CCS 2.0 系统的安装

运行 setup.exe 应用程序，弹出一个安装界面，然后选择 Code Composer Studio 项，就可以开始 CCS 2.0 的安装，按照屏幕提示可完成系统的安装。当 CCS 软件安装在计算机上之后，将在显示器桌面上出现如图 5.2 所示的两个图标。

图 5.2　CCS 设置图标

2. 系统配置

为使 CCS IDE 能工作在不同的硬件或仿真目标上，必须首先为它配置相应的配置文件。具体步骤如下：

(1) 双击桌面上的 Setup CCS 2('C 5000)图标，启动 CCS 设置。

(2) 在弹出对话框中单击“Clear”按钮，清除以前定义的配置。

(3) 从弹出的对话框中，单击“Yes”按钮，确认清除命令。

(4) 从列出的可供选择的配置文件中，选择能与使用的目标系统相匹配的配置文件。

(5) 单击加入系统配置按钮，将所选中的配置文件输入到 CCS 设置窗口当前正在创建的系统配置中，所选择的配置显示在设置窗的系统配置栏目的 My System 目录下，如图 5.3 所示。

(6) 单击“File → Save(保存)”按钮，将配置保存在系统寄存器中。

(7) 当完成 CCS 配置后，单击“File → Exit”按钮，退出 CCS Setup 。

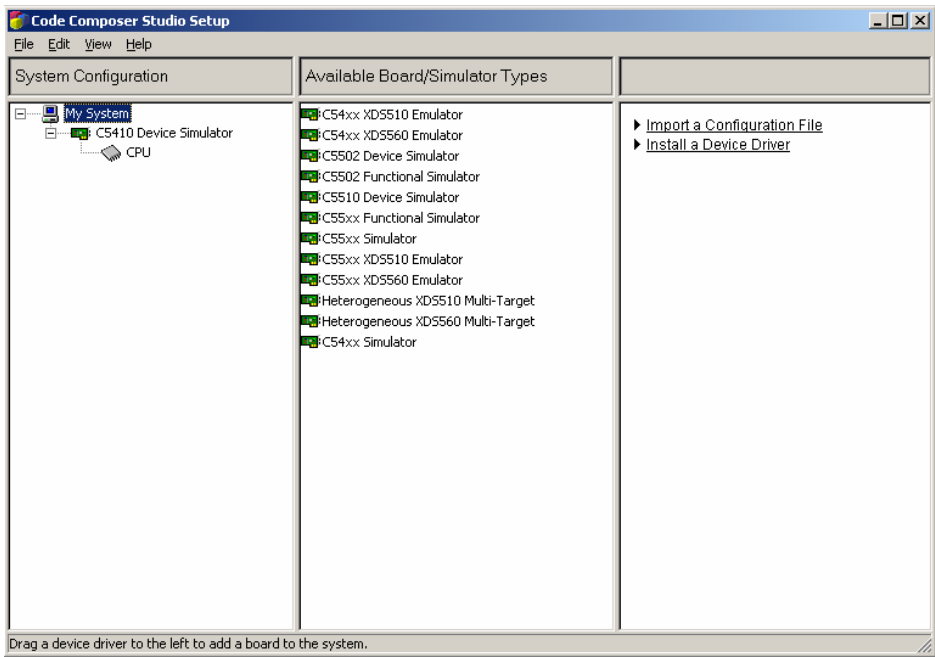

图 5.3　设置窗的系统配置栏目

3. 系统启动

双击桌面上 CCS 2('C 5000)图标，启动 CCS IDE，将自动利用刚创建的配置打开并显示 CCS 主界面。

5.1.2　CCS 的窗口、菜单和工具条

1. CCS 应用窗口

一个典型的 CCS 集成开发环境窗口如图 5.4 所示。

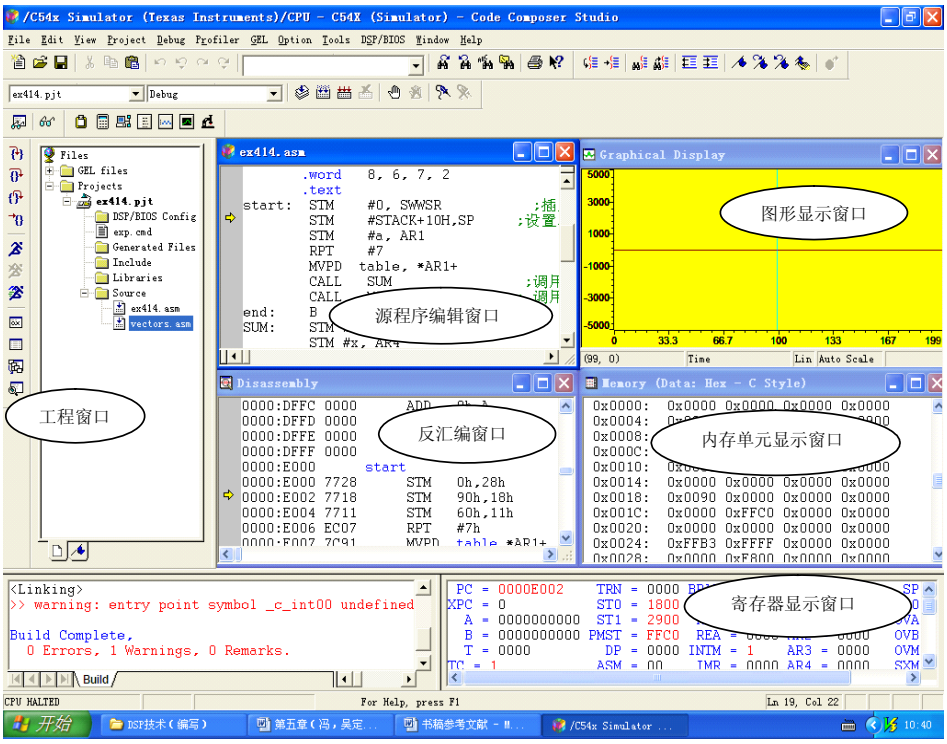

图 5.4　CCS 集成开发环境窗口

整个窗口由主菜单、工具条、工程窗口、编辑窗口、图形显示窗口、内存单元显示窗口和寄存器显示窗口等构成。

工程窗口用来组织用户的若干程序并由此构成一个项目，用户可以从工程列表中选中需要编辑和调试的特定程序。在源程序编辑窗口中，用户既可以编辑程序，又可以设置断点和探针，并调试程序。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。图形显示窗口可以根据用户需要显示数据。用户可以通过主菜单条目来管理各窗口。

2. 菜单

菜单提供了操作 CCS 的方法，由于篇幅所限这里仅就重要内容进行介绍。

1) File 菜单

File 菜单提供了与文件相关的命令，其中比较重要的操作命令如下：

(1) New→Source File　建立一个新源文件，扩展名包括*.c、*.asm、*.cmd.、*.map、*.h、*.inc、*.gel 等。

(2) New→DSP/BIOS Configuration　建立一个新的 DSP/BIOS 配置文件。

(3) New→Visual Linker Recipe　建立一个新的 Visual Linker Recipe 向导。

(4) New→ActiveX Document　在 CCS 中打开一个 ActiveX 类型的文档(如 Microsoft Excel 等)。

(5) Load Program　将 DSP 可执行的目标代码 COFF(.out)载入仿真器(Simulator 或

Emulator)中。

(6) Load GEL　加载通用扩展语言文件到 CCS 中。

(7) Data→Load　将主机文件中的数据加载到 DSP 目标系统板，可以指定存放的数据长度和地址。数据文件的格式可以是 COFF 格式，也可以是 CCS 所支持的数据格式，缺省文件格式是.dat 的文件。当打开一个文件时，会出现如图 5.5 所示的对话框。该对话框的含义是加载主机文件到数据段的从 0x0D00 处开始的长度为 0x00FF 的存储器中。

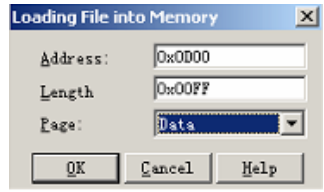

图 5.5　存储器下载对话框

(8) Data→Save　将 DSP 目标系统板上存储器中的数据加载到主机上的文件中，该命令和 Data→Load 是一个相反的过程。

(9) File I/O　允许 CCS 在主机文件和 DSP 目标系统板之间传送数据，一方面可以从 PC 机文件中取出算法文件或样本用于模拟，另一方面也可以将 DSP 目标系统处理后的数据保存在主机文件中。File I/O 功能主要与 Probe Point 配合使用。Probe Point 将告诉调试器在何时从 PC 文件中输入或输出数据。File I/O 功能并不支持实时数据交换。

2) Edit 菜单

Edit 菜单提供的是与编辑有关的命令。Edit 菜单内容比较容易理解，在这里只介绍比较重要的命令：

(1) Register　编辑指定的寄存器值，包括 CPU 寄存器和外设寄存器。由于 Simulator 不支持外设寄存器，因此不能在 Simulator 下监视和管理外设寄存器内容。

(2) Variable　修改某一变量值。

(3) Command Line　提供键入表达式或执行 GEL 函数的快捷方法。

3) View 菜单

在 View 菜单中，可以选择是否显示各种工具栏、各种窗口和各种对话框等。其中比较重要的命令如下：

(1) Disassembly　当将 DSP 可执行程序 COFF 文件载入目标系统后，CCS 将自动打开一个反汇编窗口。反汇编窗口根据存储器的内容显示反汇编指令和符号信息。

(2) Memory　显示指定存储器的内容。

(3) Registers→CPU Registers　显示 DSP 寄存器的内容。

(4) Registers→Peripheral Registers　显示外设寄存器的内容。Simulator 不支持此功能。

(5) Graph→Time/Frequency　在时域或频域显示信号波形。

(6) Graph→Constellation　使用星座图显示信号波形。

(7) Graph→Eye Diagram　使用眼图来量化信号失真度。

(8) Graph→Image　使用 Image 图来测试图像处理算法。

(9) Watch Window　用来检查和编辑变量或 C 表达式，可以以不同格式显示变量值，还可以显示数组、结构或指针等包含多个元素的变量。

(10) Call Stack　检查所调试程序的函数调用情况。此功能调试 C 程序时有效。

(11) Expression List　所有的 GEL 函数和表达式都采用表达式求值来估值。

(12) Project　CCS 启动后将自动打开视图。

(13) Mixed Source/Asm　同时显示 C 代码及相关的反汇编代码。

4) Project 菜单

CCS 使用工程(Project)来管理设计文档。CCS 不允许直接对 DSP 汇编代码或 C 语言源文件生成 DSP 可执行代码。只有建立在工程文件基础上，在菜单或工具栏上运行 Build 命令时才会生成可执行代码。工程文件被存盘为*.pjt 文件。在 Project 菜单下，除 New、Open、Close 等常见命令外，其他比较重要的命令介绍如下：

(1) Add Files to Project　CCS 根据文件的扩展名将文件添加到工程的相应子目录中。工程中支持 C 源文件(*.c*)、汇编源文件(*.a*、*.s*)、库文件(*.o*、*.lib)、头文件(*.h)和链接命令文件(*.cmd)。其中 C 和汇编源文件可以被编译和链接，库文件和链接命令文件只能被链接，CCS 会自动将头文件添加到工程中。

(2) Compile　对 C 或汇编源文件进行编译。

(3) Biuld　重新编译和链接。对那些没有修改的源文件，CCS 将不重新编译。

(4) Rebuiled All　对工程中所有文件重新编译并链接生成输出文件。

(5) Stop Build　停止正在 Build 的进程。

(6) Biuld Options　用来设定编译器、汇编器和链接器的参数。

5) Debug 菜单

Debug 菜单包含的是常用的调试命令，其中比较重要的命令介绍如下：

(1) Breakpoints　设置/取消断点命令。

程序执行到断点时将停止运行。当程序停止运行时，可检查程序的状态，查看和更改变量值，查看堆栈等。在设置断点时应注意以下两点：

① 不要将断点设置在任何延迟分支或调用指令处。

② 不要将断点设置在 repeat 块指令的倒数 1、2 行指令处。

(2) Probe Points　探测点设置。

允许更新观察窗口并在设置 Probe Points 处将 PC 文件数据读至存储器或将存储器数据写入 PC 文件，此时应设置 File I/O 属性。

对每一个建立的窗口，默认情况是在每个断点(Breakpoints)处更新窗口显示，然而也可以将其设置为到达 Probe Points 处更新窗口。使用 Probe Points 更新窗口时，目标 DSP 将临时中止运行，当窗口更新后，程序继续运行。因此 Probe Points 不能满足实时数据交换(RTDX)的需要。

(3) StepInto　单步运行。如果运行到调用函数处将跳入函数单步运行。

(4) StepOver　执行一条 C 指令或汇编指令。与 StepInto 不同的是，为保护处理器流水线，该指令后的若干条延迟分支或调用将同时被执行。如果运行到函数调用处将执行完该函数而不跳入函数执行，除非在函数内部设置了断点。

(5) StepOut　如果程序运行在一个子程序中，执行 StepOut 将使程序执行完该子程序后回到调用该函数的地方。在 C 源程序模式下，根据标准运行 C 堆栈来推断返回地址，否则根据堆栈顶的值来求得调用函数的返回地址。因此，如果汇编程序使用堆栈来存储其他信息，则 StepOut 命令可能工作不正常。

(6) Run　当前程序计数器(PC)执行程序，碰到断点时程序暂停运行。

(7) Halt　中止程序运行。

(8) Animate　动画运行程序。当碰到断点时程序暂时停止运行，在更新未与任何 Probe

Points 相关联的窗口后程序继续执行。该命令的作用是在每个断点处显示处理器的状态，可以在 Option 菜单下选择 Animate Speed 来控制其速度。

(9) Run Free　忽略所有断点(包括 Probe Points 和 Profile Points)，从当前 PC 处开始执行程序。此命令在 Simulator 下无效。使用 Emulator 进行仿真时，此命令将断开与目标 DSP 的连接，因此可移走 JTAG 和 MPSD 电缆。在 Run Free 时还可对目标 DSP 硬件复位。

(10) Run to Cursor　执行到光标处，光标所在行必须为有效代码行。

(11) Multiple Operation　设置单步执行的次数。

(12) Reset DSP　复位 DSP，初始化所有寄存器到其上电状态并中止程序运行。

(13) Restart　将 PC 值恢复到程序的入口。此命令并不开始程序的运行。

(14) Go Main　在程序的 main 符号处设置一个临时断点。此命令在调试 C 程序时起作用。

6) Profiler 菜单

剖切点(profiler points)是 CCS 的一个重要的功能，它可以在调试程序时，统计某一块程序执行所需要的 CPU 时钟周期数、程序分支数、子程序被调用数和中断发生次数等统计信息。Profile Point 和 Profile Clock 作为统计代码执行的两种机制，常常一起配合使用。Profiler 菜单的主要命令介绍如下：

(1) Enable Clock　使能剖析时钟。

为获得指令的周期及其他事件的统计数据，必须使能剖析时钟(profile clock)。当剖析时钟被禁止时，将只能计算到达每个剖析点的次数，而不能计算统计数据。

指令周期的计算方式与 DSP 的驱动程序有关，对使用 JTAG 扫描路径进行通信的驱动程序，指令周期通过处理器的片内分析功能进行计算，其他驱动程序则可以使用其他类型的定时器。Simulator 使用模拟的 DSP 片内分析接口来统计剖析数据。当时钟使能时，CCS 调试器将占用必要的资源以实现指令周期的计算。

剖析时钟作为一个变量(CLK)通过 Clock 窗口被访问。CLK 变量可在 Watch 窗口观察，并可在 Edit Variable 对话框修改其值。CLK 还可以在用户定义的 GEL 函数中使用。

Instruction Cycle Time 用于执行一条指令的时间，其作用是在显示统计数据时将指令周期数转化成时间或频率。

(2) Clock Setup　时钟设置。单击该命令将出现如图 5.6 所示的 Clock Setup 对话框。

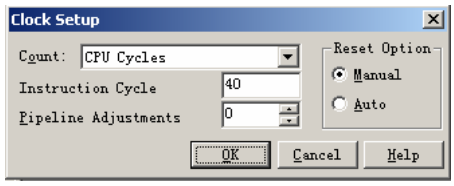

图 5.6　Clock Setup 对话框

在 Count 域内选择剖析的事件。使用 Reset Option 参数可以决定如何计算。如选择 Manual 选项，则 CLK 变量将不断累计指令周期数；如选择 Auto 选项，则在每次 DSP 运行前自动将 CLK 设置为 0。因此，CLK 变量显示的是上一次运行以来的指令周期数。

(3) View Clock　打开 Clock 窗口，以显示 CLK 变量的值。

双击 Clock 窗口的内容可直接复位 CLK 变量(使 Clock=0)。

7) Option 菜单

Option 菜单提供 CCS 的一些设置选项，其中比较重要的命令介绍如下：

(1) Font　设置字体。该命令可以设置字体、大小及显示样式等。

(2) Disassembly Style Options　设置反汇编窗口显示模式，包括反汇编成助记符或代数符号，直接寻址与间接寻址，用十进制、二进制或十六进制显示。

(3) Memory Map　用来定义存储器映射。存储器映射指明了 CCS 调试器不能访问哪段存储器。典型情况下，存储器映射与命令文件的存储器定义一致。

8) GEL 菜单

CCS 软件本身提供了 C54X 和 C55X 的 GEL 函数，它们在 c5000.gel 文件中定义。GEL 菜单中包括 CPU_Reset 和 C54X_Init 命令。

(1) CPU_Reset　该命令复位目标 DSP 系统、复位存储器映射(处于禁止状态)以及初始化寄存器。

(2) C54X_Init　该命令也对目标 DSP 系统复位，与 CPU_Reset 命令不同的是，该命令使能存储器映射，同时复位外设和初始化寄存器。

9) Tools 菜单

Tools 菜单提供了常用的工具集，这里就不再介绍了。

3. 工具栏

CCS 集成开发环境提供 5 种工具栏，以便执行各种菜单上相应的命令。这 5 种工具栏可在 View 菜单下选择是否显示。

(1) Standard Toolbar(标准工具栏)，如图 5.7 所示，包括新建、打开、保存、剪切、复制、粘贴、取消、恢复、查找、打印和帮助等常用工具。

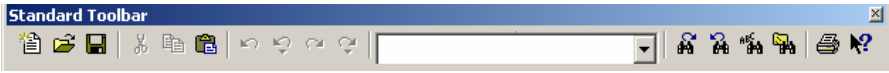

图 5.7　标准工具栏

(2) Project Toolbar(工程工具栏)，如图 5.8 所示，包括选择当前工程、编译文件、设置和移去断点、设置和移去 Probe Point 等功能。

图 5.8　工程工具栏

(3) Edit Toolbar，提供了一些常用的查找和设置标签命令，如图 5.9 所示。

图 5.9　Edit 工具栏

(4) GEL Toolbar，提供了执行 GEL 函数的一种快捷方法，如图 5.10 所示。在工具栏左侧的文本输入框中键入 GEL 函数，再单击右侧的执行按钮即可执行相应的函数。如果不使用 GEL 工具栏，也可以使用 Edit 菜单下的 Edit Command Line 命令执行 GEL 函数。

(5) ASM/Source Stepping Toolbar ，提供了单步调试 C 或汇编源程序的方法，如图 5.11 所示。

图 5.10　GEL 工具栏

图 5.11　ASM/Source Stepping 工具栏

(6) Target Control Toolbar，提供了目标程序控制的一些工具，如图 5.12 所示。

(7) Debug Window Toolbar，提供了调试窗口工具，如图 5.13 所示。

图 5.12　Target Control 工具栏

图 5.13　Debug Window 工具栏

5.1.3　CCS 工程管理

CCS 对程序采用工程(Project)的集成管理方法。工程保持并跟踪在生成目标程序或库过程中的所有信息。一个工程包括以下的内容：

- 源代码的文件名和目标库的名称；
- 编译器、汇编器、连接器选项；
- 有关的包括文件。

本节说明在 CCS 中如何创建和管理用户程序。

1. 工程的创建、打开和关闭

每个工程的信息存储在单个工程文件(*.pjt)中。可按以下步骤创建、打开和关闭工程。

1) 创建一个新工程

选择“Project → New(工程 → 新工程)”，如图 5.14 所示，在 Project 栏中输入工程名字，其他栏目可根据习惯设置。工程文件的扩展名是*.pjt。若要创建多个工程，每个工程的文件名必须是唯一的。但可以同时打开多个工程。

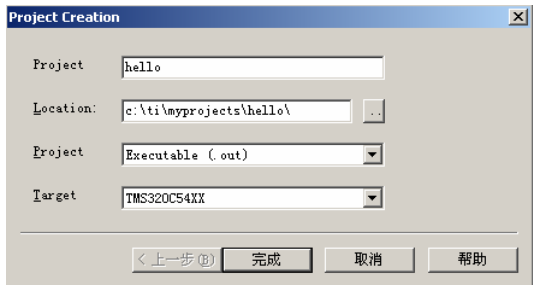

图 5.14　建立新工程对话框

2) 打开已有的工程

选择“Project → Open(工程 → 打开)”，弹出如图 5.15 所示工程打开对话框。双击需要打开的文件(*.pjt)即可。

3) 关闭工程

选择“Project→Close(工程→关闭)”，即可当前关闭工程。

2. 使用工程观察窗口

工程窗口图形显示工程的内容。当打开工程时，工程观察窗口自动打开如图 5.16 所示。要展开或压缩工程清单，单击工程文件夹、工程名(*.pjt)和各个文件夹上的“+/−”号即可。

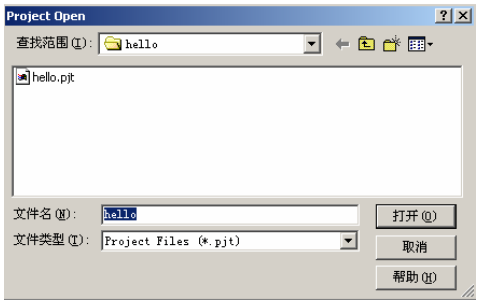

图 5.15　打开工程对话框

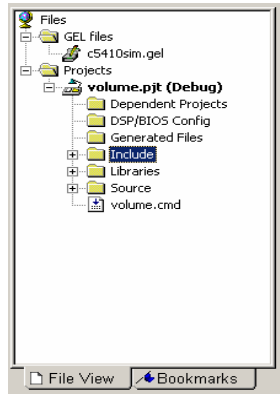

图 5.16　工程观察窗口

3. 加文件到工程

可按以下步骤将与该工程有关的源代码、目标文件、库文件等加入到工程清单中去。

1) 加文件到工程

(1) 选择“Project→Add Files to Project(工程→加文件到工程)”，出现 Add Files to Project 对话框。

(2) 在 Add Files to Project 对话框，指定要加入的文件。如果文件不在当前目录中，浏览并找到该文件。

(3) 单击“打开”按钮，将指定的文件加到工程中去。当文件加入时，工程观察窗口将自动的更新。

2) 从工程中删除文件

(1) 按需要展开工程清单。

(2) 右击要删除的文件名。

(3) 从上下文菜单，选择“Remove from Project(从工程中删除)”。

在操作过程中，注意文件扩展名，因为文件通过其扩展名来辨识。

5.1.4　CCS 源文件管理

1. 创建新的源文件

可按照以下步骤创建新的源文件：

(1) 选择“File→New→Source File(文件→新文件→源文件)”，将打开一个新的源文件编辑窗口。

(2) 在新的源代码编辑窗口输入代码。

(3) 选择“File→Save(文件→保存)”或“File→Save As(文件→另存为)”，保存文件。

2. 打开文件

可以在编辑窗口打开任何 ASCII 文件。

(1) 选择“File→Open(文件→打开)”，将出现如图 5.17 所示打开文件对话框。

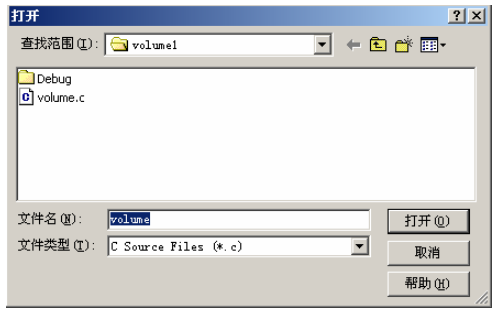

图 5.17　打开文件对话框

(2) 在打开文件对话框中双击需要打开的文件，或者选择需要打开的文件，并单击“打开”按钮。

3. 保存文件

(1) 单击编辑窗口，激活需要保存的文件。

(2) 选择“File→Save(文件→保存)”，输入要求保存的文件名。

(3) 在保存类型栏中，选择需要的文件类型，如图 5.18 所示。

(4) 单击“保存”按钮。

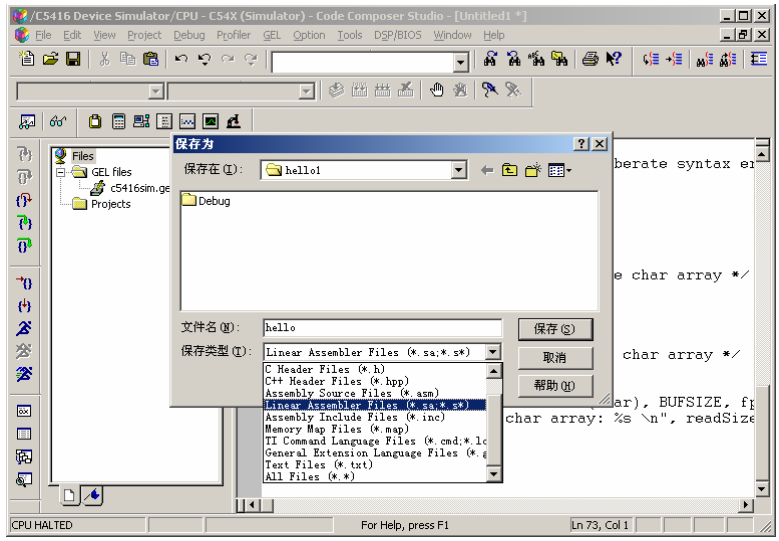

图 5.18　保存文件对话框

5.1.5　通用扩展语言 GEL

通用扩展语言 GEL(General Extension Language)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，特别适合用于自动测试和自定义工作空间。关于 GEL 详细内容参见 TI 公司的《TMS320C54x Code Composer Studio User's Guide》手册。

5.2　CCS 应用举例

本节讲述开发一个具备基本信号处理功能的 DSP 程序的过程。首先介绍如何创建一个工程、向工程中添加源文件、浏览代码、编译和运行程序、修改 Build 选项并更正语法错误、使用断点和 Watch 窗口等基本应用；其次介绍使用探针和图形显示的方法。

5.2.1　基本应用

1. 创建一个工程

(1) 选择“Project→New(工程→新建)”，弹出工程建立对话框。

(2) 在 Project 栏输入文件名 Volume。默认的工作目录是 C:\ti\myprojects\(假设 CCS 安装在 C:\ti 下)，其他两项也选默认即可。

(3) 单击完成按钮，将在工程窗口的 Project 下面创建 Volume 工程。

2. 向工程中添加源文件

(1) 将“C:\ti\tutorial\sim54xx\Volume1”(假设 CCS 安装在 C:\ti 下)下全部文件复制到新建的“C:\ti\myprojects\Volume”目录下。

(2) 选择“Project→Add Files to Project(工程→加载文件)”，在文件加载对话框中选择 Volume.c 文件，单击“打开”按钮将 Volume.c 添加到工程中，如图 5.19 所示。

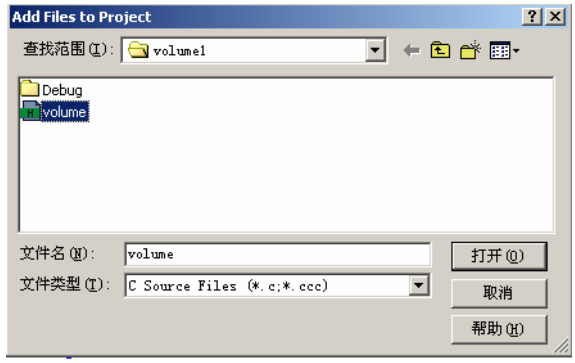

图 5.19　添加 Volume.c 文件

(3) 用同样方法将 Vector.asm 添加到工程中。Vector.asm 中包含的是将 RESET 中断指向 C 程序入口 c_int00 的汇编指令和其他中断的入口指令。如果调试的程序较为复杂，则

可在 Vector.asm 中定义更多的中断矢量。

(4) 将 Volume.cmd 添加到工程文件中。该文件的作用是将段(Sections)分配到存储器中。

(5) 将 load.asm 添加到工程文件中。该文件包含一个简单的汇编循环程序，被 C 程序调用。调用时带有一个参数(argument)，执行此程序共需约 1000×argument 个指令周期。

(6) 将“C:\ti\c5400\cgtools\lib”下的 rts.lib 加入到工程文件中。该文件是采用 C 语言开发 DSP 应用程序的运行支持库函数。

在工程中双击所有“+”，即可看到整个工程的文件。在以上的操作中，没有将头文件加到工程中，CCS 将在 Bulid 时自动查找所需的头文件。

3. 浏览代码

双击 Project 视图中的 Volume.c，将在代码窗口看到源文件代码。

```
#include <stdio.h>
#include “volume.h”
/* Global declarations */
int inp_buffer[BUFSIZE];                /* processing data buffers */
int out_buffer[BUFSIZE];
int gain = MINGAIN;                     /* volume control variable */
unsigned int processingLoad = BASELOAD; /* processing routine load value
*/
struct PARMS str =
{
   2934,
   9432,
   213,
   9432,
   &str
};
/* Functions */
extern void load(unsigned int loadValue);
static int processing(int *input, int *output);
static void dataIO(void);
/*
 * ======== main ========
 */
void main()
{
   int *input = &inp_buffer[0];
   int *output = &out_buffer[0];
   puts("volume example started\n");
   /* loop forever */
   while(TRUE)
   {
```

```
        /*
         *  Read input data using a probe-point connected to a host file.
         *  Write output data to a graph connected through a probe-point.
         */
        dataIO();
        #ifdef FILEIO
        puts("begin processing")       /* deliberate syntax error */
        #endif
    /* apply gain */
        processing(input, output);
    }
}
/*
 *  ======== processing ========
 *
 * FUNCTION: apply signal processing transform to input signal.
 *
 * PARAMETERS: address of input and output buffers.
 *
 * RETURN VALUE: TRUE.
 */
static int processing(int *input, int *output)
{
    int size = BUFSIZE;
    while(size--){
        *output++ = *input++ * gain;
    }
    /* additional processing load */
    load(processingLoad);
    return(TRUE);
}
/*
 *  ======== dataIO ========
 *
 * FUNCTION: read input signal and write processed output signal.
 *
 * PARAMETERS: none.
 *
 * RETURN VALUE: none.
 */
static void dataIO()
{
    /* do data I/O */
    return;
}
```

从以上代码可以看出：

(1) 主程序显示一条提示信息后，进入一个无限循环，不断调用 dataIO 和 processing 两个函数。

(2) processing 函数将输入 buffer 中的数与增益相乘，并将结果输出给 buffer，它还调用汇编 load 例程的参数 processingLoad 的值计算指令周期的时间。

(3) dataIO 函数不执行任何实质操作。它没有使用 C 代码执行 I/O 操作，而是通过 CCS 中的 Probe Point 工具，从 PC 机文件中读取数据到 inp_buffer 中，作为 processing 函数的输入参数。

4. 编译和运行程序

(1) 选择“Project→Rebuild All(工程→重新编译)”，对工程进行重新编译。

(2) 选择“File→Load Program(文件→下载程序)”，选 volume.out 并打开，将 Build 生成的程序加载到 DSP。

(3) 选择“View→Mixed Source/ASM(查看→混合 C 程序/汇编)”。该设置使得 C 程序与其汇编结果同时显示。

(4) 在反汇编窗口中单击汇编指令，按 F1 键切换到在线帮助窗口，显示光标所在行的关键词的帮助信息。

(5) 选择“Debug→Go Main(调试→到主程序首)”来使得程序从主程序开始执行。

(6) 选择“Debug→Run(调试→运行)”，可以在 Output 窗口看到“begin processing”信息。

(7) 选择“Debug→Halt(调试→停止)，中止正在执行的程序。

5. 修改 Build 选项并更正语法错误

在以上的程序中由于 FILEIO 没有定义，因而在编译时将忽略程序中的部分代码，这样在链接生成的 DSP 程序中也不包括这部分代码。下面通过更改程序选项来定义 FILEIO，从而将这部分代码生成到执行程序中。

(1) 选择“Project→Build Options(工程→编译选项)”。

(2) 在 Compiler 栏的 Categroy 域，单击 Preprocessor。在右侧的 Define Symbols 中键入 FILEIO。这时将在编译参数栏中看到“-dFILEIO”，如图 5.20 所示。在定义 FILEIO 后，C 编译器将对所有的源代码进行编译。

(3) 单击“确定”按钮，保存选项设置结果。

(4) 选择“Project→Rebuild All(工程→重新编译)”。在工程选项更改后，重新编译程序是必须的。

(5) 此时 output 窗口提示源代码中存在语法错误，错误出现在第 68 行，如图 5.21 所示。在该行后加分号再存盘，重新编译程序并生成新的 volume.out 文件。

6. 使用断点和 Watch 窗口

(1) 选择“File→Reload Program(文件→重新下载程序)”，重新下载程序。

(2) 在工程视图中双击 volume.c，打开源文件编辑窗口。

(3) 将光标放在“dataIO();”行。

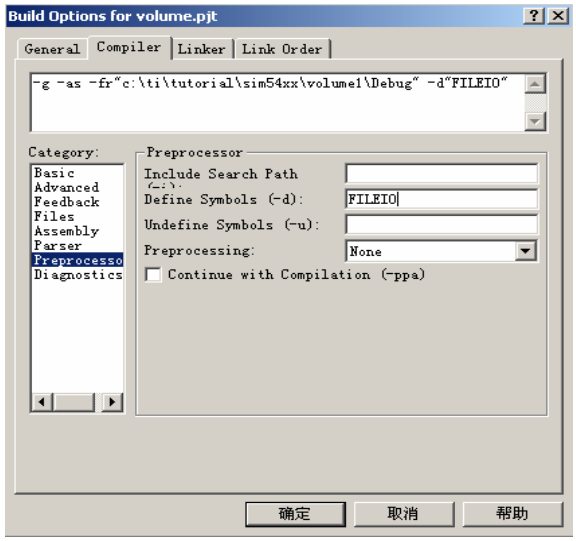

图 5.20　在 Build Options 下定义 FILEIO

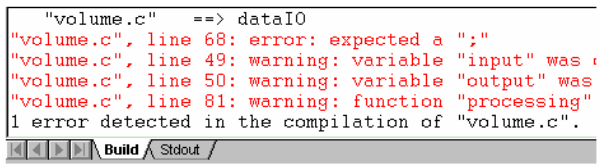

图 5.21　编译错误提示

(4) 单击鼠标右键，在弹出菜单上选择 Toggle breakpoint，设置断点。

(5) 选择“View→Watch Window(查看→观察窗口)”，将出现 Watch 窗口。程序运行时 Watch Window 窗口将显示要查看的变量值。

(6) 选择 Watch1 栏。

(7) 在 Watch1 窗口单击图标，在 name 栏输入 dataIO。

(8) 选择“Debug→Go Main(调试→到主程序首)”。

(9) 选择“Debug→Run(调试→运行)”，运行程序，如图 5.22 所示。显示出 dataIO 是一个函数，该函数存放的首地址是 0x00001457。

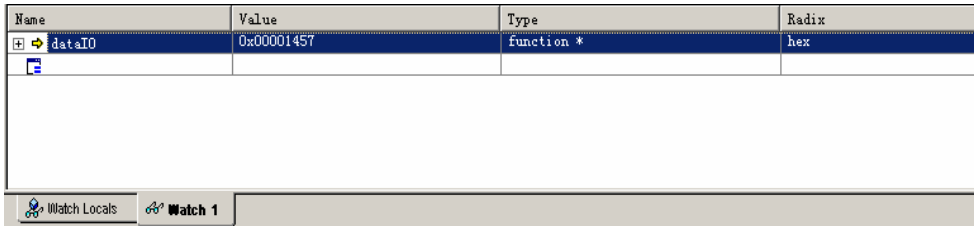

图 5.22　运行后的 Watch1 窗口

7. 使用 Watch 窗口观察结构体

仿照上面的方法，在 Watch 窗口中加入 str 结构体变量。可以看到在 str 左边有一个“+”

标志，表明 str 是一个结构体。双击“+”后将看到 str 结构体中包含的元素，如图 5.23 所示，双击每个元素可以更改其数值大小。

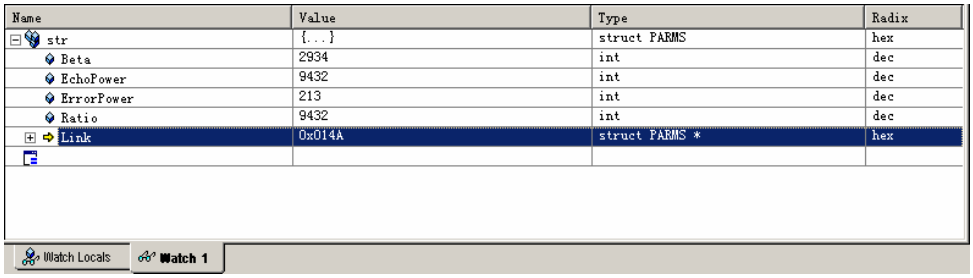

图 5.23　Watch 窗口的结构体显示

在 Watch 窗口中单击右键，在弹出菜单时还可选择：移去一个表达式、隐藏 Watch 窗口等。可以通过选择“Debug→Breakpoints(调试→断点)”，在该窗口中单击 Delete All 按钮将所有断点去掉。

5.2.2　探针和显示图形的使用

本实例介绍创建和测试一个简单数字信号算法的过程，所需处理的数据放在 PC 文件中。通过本实例学习使用探针和图形显示的方法。

Probe Point 是开发算法的一个有用工具，可以使用 Probe Point 从 PC 文件中存取数据。即

- 将 PC 文件中数据传送到目标板上的 buffer，供算法使用。
- 将目标板上 buffer 中的输出数据传送到 PC 文件中以供分析。
- 更新一个窗口，如由数据绘出的 Graph 窗口。

Probe Point 与 Breakpoints 都会中断程序的运行，但 Probe Point 与 Breakpoints 在以下方面不同。Probe Point 只是暂时中断程序运行，当程序运行到 Probe Point 时会更新与之相连接的窗口，然后自动继续运行程序；Breakpoints 中断程序运行后，将更新所有打开的窗口，且只能用人工的方法恢复程序运行；Probe Point 可与 FILEIO 配合，在目标板与 PC 文件之间传送数据，Breakpoints 则无此功能。

下面讲述如何使用 Probe Point 将 PC 文件中的内容作为测试数据传送到目标板。同时使用一个断点以便在到达 Probe Point 时自动更新所有打开的窗口。

1. 为 FILE I/O 添加 Probe Point

(1) 打开 5.2 节已经完成的程序，并进行编译。

(2) 选择“File→Load Program(文件→下载程序)”。选择 volume1.out 文件，并单击打开按钮。

(3) 双击 volume.c，以便在右边的编辑窗口显示源代码。

(4) 将光标放在主函数的 dataIO()行上。

(5) 单击鼠标右键，在弹出菜单中选择“Toggle Probe Point”，添加 Probe Point。

(6) 在 File(文件)菜单，选择“File I/O”，出现 File I/O 对话框，如图 5.24 所示，在对话框中选择输入/输出文件。

图 5.24　File I/O 对话框

(7) 在“File Input”栏中，单击 Add File 按钮。

(8) 在 volume.c 文件所在目录选择 sine.dat，并单击打开按钮。此时将出现一个控制窗口，如图 5.25 所示。可以在运行程序时使用这个窗口来控制数据文件的开始、停止、前进、后退等操作。

图 5.25　File I/O 控制窗口

(9) 在 File I/O 对话框中，在 Address 域填入 inp_buffer，在 length 域填入 100，同时选中 Wrap Around 复选框(如图 5.26 所示)。这几部分值含义如下：

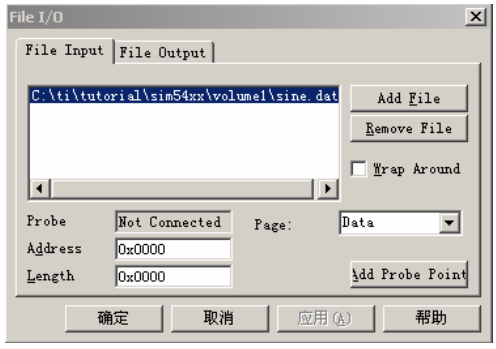

图 5.26　File I/O 属性

① Address 域　从文件中读取的数据将要存放的地址。inp_buffer 是在 volume.c 中定义的整型数组，其长度为 BUFFSIZE。

② Length 域　每次到达 Probe Point 时从数据文件中读取多少个样点。这里取值为 100 是因为 BUFFSIZE=100，即每次取 100 个样值放在输入缓冲中。如果 Length 超过 100 则可能导致数据丢失。

③ Wrap Around 复选框　表明读取数据的循环特性，每次读至文件结尾处将自动从文件头开始重新读取数据。这样将从数据文件中读取一个连续(周期性)的数据流。

(10) 单击“Add Probe Point”按钮，将出现 Break/Probe Points 对话框，如图 5.27 所示，选中“Probe Points”栏。

(11) 在 Probe Point 列表中显示“VOLUME.C line 61 --> No Connection”。表明该第 61 行已经设置 Probe Point，但还没有和 PC 文件关联。

(12) 在 Connect 域，单击向下箭头并从列表中选 sine.dat。

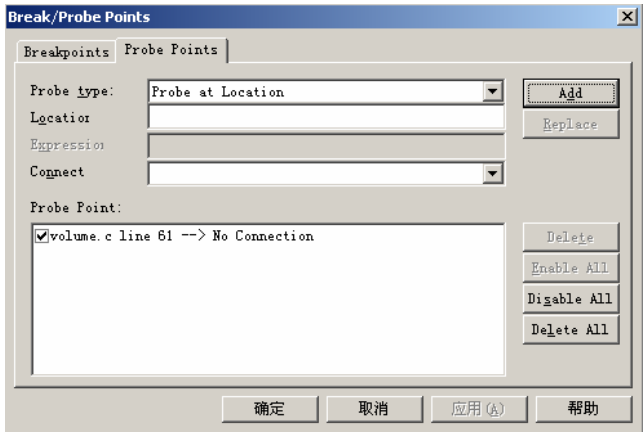

图 5.27　Break/Probe Points 对话框

(13) 单击 Replace 按钮，Probe Point 列表框表示 Probe Point 已与 sine.dat 文件相关联。

(14) 单击“确定”按钮，File I/O 对话框指示文件连至一个 Probe Point。

(15) 单击“确定”按钮，关闭 File I/O 对话框。

2. 显示图形

如果现在运行程序，将看不到任何程序运行结果。当然可以设置 Watch 窗口观察 inp_buffer 和 out_buffer 等的值，但需要观察的变量很多，而且显示的也只是枯燥的数据，远不如图形显示直观、友好。

CCS 提供很多方法将程序产生的数据以图形显示，包括时域/频域波形、星座图、眼图等。在本例中使用时域/频域波形显示功能观察一个时域波形。

(1) 选择“View→Graph→Time/Frequency(显示→图形→时域/频域)”。弹出 Graph Property 对话框，如图 5.28 所示。

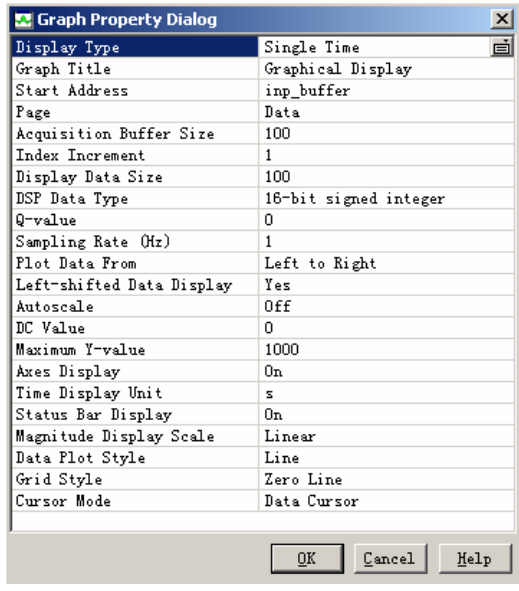

图 5.28　更改图形属性

(2) 在 Graph Property 对话框中，更改 Graph Title(图形标题)、Start Address(起始地址)、Acquisition BufferSize(采集缓冲区大小)、DSP Data Type(DSP 数据类型)、Autoscale(自动伸缩属性)及 Maximum Y-value(最大 Y 值)。

(3) 单击 OK 按钮，将出现一个显示 inp_buffer 波形的图形窗口。

(4) 在图形窗口中右击，从弹出菜单中选择 Clear Display，清除已显示波形。

(5) 再次执行“View→Graph→Time/Frequency”。

(6) 将 Graph Title 修改为 output buffer，Start Address 修改为 out_buffer，其他设置不变。

(7) 单击 OK 按钮，出现一个显示 out_buffer 波形的图形窗口，右击从菜单中选择 Clear Display 命令，清除已有显示波形。

3. 动态显示程序和图形

到现在为止，已经设置了一个 Probe Point。它将临时中断程序运行，将 PC 上数据传给目标板，然后继续运行程序。但是，Probe Point 不会更新图形显示内容。本节将设置一个断点，使图形窗口自动更新。使用 Animate 命令，使程序到达断点时更新窗口后自动继续运行。

(1) 在 volume.c 窗口，将光标放在 dataIO 行上。

(2) 在该行上同时设置一个断点和一个 Probe Point，这使得程序在只中断一次的情况下执行两个操作：传送数据和更新图形显示。

(3) 重新组织窗口以便能同时看到两个图形窗口。

(4) 在 Debug 菜单单击 Animate。此命令将运行程序，碰到断点后临时中断程序运行，更新窗口显示，然后继续执行程序。与 Run 不同的是，Animate 会继续执行程序直到碰到下一个断点。只有人为干预时，程序才会真正中止运行。可以将 Animate 命令理解为一个“运行→中断→继续”的操作。

(5) 每次碰到 Probe Point 时，CCS 将从 sine.dat 文件读取 100 个样值，并将其写至输入缓冲 inp_buffer。由于 sine.dat 文件保存的是 40 个采样值的正弦波形数据，因此每个波形包括 2.5 个 sin 周期波形，如图 5.29 所示。

(6) 选择“Debug→Halt(调试→停止)”，停止程序运行。

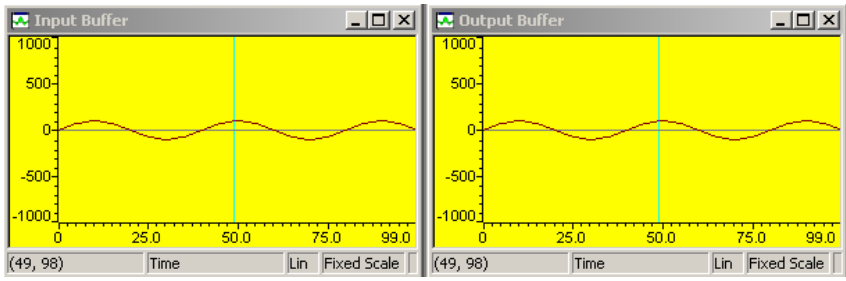

图 5.29　Gain＝1 时的输入/输出图形显示

4. 增益调节

本程序将输入缓冲的数据与增益相乘后送至输出缓冲中：

```
output++=input++*gain
```

增益被初始化为 MINGAIN，在 volume.h 中定义为 1。为改变输出值，需改变增益，方法之一是使用 Watch 功能。

(1) 选择“View→Watch Window(查看→观察窗口)”。

(2) 在 Watch 窗口右击，选择“Insert New Expression”。

(3) 输入 Gain 作为要观察的表达式，单击 OK 按钮。

(4) 如程序已中止运行，单击 Animate 按钮重新运行程序。

(5) 在 Watch 窗口双击 Gain。

(6) 在变量编辑窗口将 Gain 值改为 10，单击 OK 按钮。

注意到输出缓冲图中的幅度值已经变为原来的 10 倍，如图 5.30 所示。

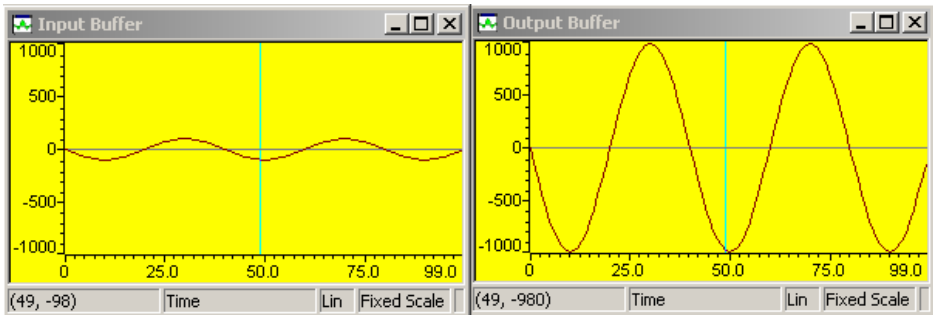

图 5.30　Gain=10 时的输入/输出图形显示

5.3　CCS 仿真

5.3.1　用 Simulator 仿真中断

C54x 允许用户仿真外部中断信号 INT0～INT3，并选择中断发生的时钟周期。为此，可以建立一个数据文件，并将其连接到 4 个中断引脚中的一个即 INT0～INT3，或 BIO 引脚。值得注意的是时间间隔用 CPU 时钟周期函数来表示，仿真从一个时钟周期开始。

1. 设置输入文件

为了仿真中断，必须先设置一个输入文件(输入文件使用文本编辑器编辑)，列出中断间隔。文件中必须有如下格式的时钟周期

```
[clock  clock，logic  value]rpt {n  |EOS}
```

只有使用 BIO 引脚的逻辑时，才使用方括号。

(1) clock　clock(时钟周期)是指希望中断发生时的 CPU 时钟周期。可以使用两种 CPU 时钟周期。

① 绝对时钟周期是指其周期值表示所要仿真中断的实际 CPU 时钟周期。

如 14、26、58。分别表示在第 14、26、58 个 CPU 时钟周期处仿真中断，对时钟周期值没有操作，中断在所写的时钟周期处发生。

② 相对时钟周期是指相对于上次事件的时钟周期。

如 14+26 和 58。表示有 3 个时钟周期，即分别在 14、40(14＋26)和 58 个 CPU 时钟周期处仿真中断。时钟周期前面的加号表示将其值加上前面总的时钟周期。在输入文件中可以混合使用绝对时钟周期和相对时钟周期。

(2) logic value(逻辑值)只使用于 BIO 引脚。必须使用一个值去迫使信号在相应的时钟周期处置高位和置低位。

如[13，1]、[25，0]和[55，1]表示 BIO 在第 13 个时钟周期置高位，在第 25 时钟周期置低位，在第 55 时钟周期又置高位。

(3) rpt {n　|EOS}是一个可选参数，代表一个循环修正。可以用两种循环形式来仿真中断：

① 固定次数的仿真。可以将输入文件格式化为一个特定模式并重复一个固定次数

如 5(+10 +20)rpt 2。括号中的内容代表要循环的部分，这样在第 5 个 CPU 时钟周期仿真一个中断，然后在第 15(5+10)、35(15+20)、45(35+10)、65(45+15)个时钟周期处仿真一个中断。n 是一个正整数，表示重复循环的次数。

② 循环直到仿真结束。为了将同样模式在整个仿真过程中循环，加上一个 EOS。

如 5(+10 +20)rpt EOS 表示在第 5 个 CPU 时钟周期仿真一个中断，然后在第 15(5+10)、35(15+20)、45(35+10)、65(45+15)个时钟周期处仿真一个中断，并将该模式持续到仿真结束。

2. 软件仿真编程

建立输入文件后，就可以使用 CCS 提供的 Tools→Pin connect 菜单来连接列表及将输入文件与中断脚断开。使用调试单击 Tools→Command Window，系统出现如图 5.31 所示的窗口。

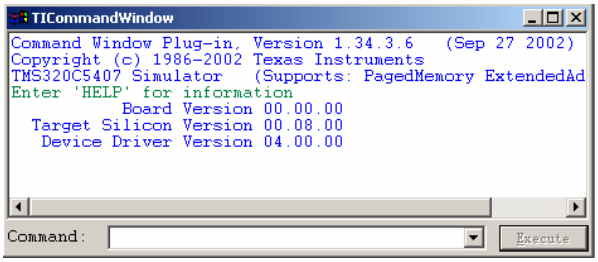

图 5.31　Command Window 窗口

在输入窗口的 Command 处根据需要选择输入如下命令。

(1) pinc　将输入文件和引脚相连。

命令格式：pinc 引脚名，文件名。

引脚名：确认引脚必须是 4 个仿真引脚(INT0～INT3)中的一个，或是 BIO 引脚。

文件名：输入文件名。

(2) pinl　验证输入文件是否连接到了正确的引脚上。

命令格式：pinl。

它首先显示所有没有连接的引脚，然后是已经连接的引脚。对于已经连接的引脚，在 Command 窗口显示引脚名和文件的绝对路径名。

(3) pind　结束中断，引脚脱开。

命令格式：pind 引脚名。

该命令将文件从引脚上脱开，则可以在该引脚上连接其他文件。

3. 实例

Simulator 仿真 INT3 中断，当中断信号到来时，中断处理子程序完成将一变量存储到数据存储区中，中断信号产生 10 次。

(1) 编写中断产生文件。设置输入文件，列出中断发生间隔。在文件 zhongduan.txt 中写入 100(+100)rpt 10 之后存盘，此文件与中断的 INT3 引脚连接后，系统每隔 100 个时钟周期发生一次中断。

(2) 将输入文件 zhongduan.txt 连接到中断引脚。在命令行输入 pinc INT3，zhongduan.txt，将 INT3 引脚与 zhongduan.txt 文件连接。

(3) 用汇编语言仿真中断。

① 编写中断向量表。对于要使用的中断引脚，应正确的配置中断入口和中断服务子程序。在源程序的中断向量表中写入：

```
    .mmregs
    ; 建立中断向量
    .sect "vectors"
    .space 93*16          ; 在中断向量表中预留一定空间，使程序能够正确转移
    INT3                  ; 外部中断 INT3
    NOP
    NOP
    GOTO NT3
    NOP
    .space 28*16          ; 68h～7Fh 保留区
```

② 编写主程序。在主程序中，要对中断有关的寄存器进行初始化。

```
    ***********zhongduansim***********
    .data
a0  .word  0,0,0,0,0,0,0,0
    .text
    .global  _main
_main:
    PMST = #01a0h         ; 初始化 PMST 寄存器
    SP=#27FFh             ; 初始化 SP 寄存器
    DP=#0
    IMR=#100              ; 初始化 IMR 寄存器
    AR1=#a0
    a=#9611h
    INTM=0                ; 开中断
wait                      ; 等待中断信号
    NOP
    NOP
```

```
    GOTO wait
```

③ 编写中断服务程序。

```
NT3:
    NOP
    NOP
    (*AR1+)=a;
    NOP
    NOP
    return_enable
    .end
```

在命令窗口输入 reset，然后装入编译和连接好的*.out 程序，开始运行。

5.3.2　用 Simulator 仿真 I/O 端口

用 Simulator 仿真 I/O 端口，可按如下 3 个步骤实现：

- 定义存储器映射方法；
- 连接 I/O 端口；
- 脱开 I/O 端口。

实现这些步骤可以使用系统提供的 Tools→Port Connect 菜单来连接、脱开 I/O 端口，也可以选择调试命令来实现。单击 Tools→Command Window，系统将弹出对话框，然后在 Command 处根据需要选择输入的命令。

1. 定义存储器映射方法

定义存储器映射除了前面章节讲的方法以外，还可以在命令窗口输入 ma 命令定义实际的目标存储区域，语法如下

```
ma  address,page,length,type
```

(1) address　定义一个存储区域的起始地址，此参数可以是一个绝对地址、C 表达式、函数名或汇编语言标号。

(2) page　用来识别存储器类型，0 代表程序存储器，1 代表数据存储器，2 代表 I/O 空间。

(3) length　定义其长度，可以是任何 C 表达式。

(4) type　说明该存储器的读写类型。该类型必须是表 5-1 关键字中的一个。

表 5-1　存储器读写类型对应的关键字

存储器类型	type 类型
只读存储器	R 或 ROM
只写存储器	W 或 WOM
读写存储器	R|M 或 RAM
读写外部存储器	RAM|EX 或 R|M|EX
只读外部结构	P|R
读写外部结构	P|R|W

2. 连接 I/O 端口

mc(memory connect)将 P|R，PW，P|R|W 连接到输入/输出文件。允许将数据区的任何区域(除 00H～1FH)连接到输入/输出文件来读写数据。语法如下

```
mc portaddress,page,length,flename,fileaccess
```

(1) portaddress　I/O 空间或数据存储器地址。此参数可以是一个绝对地址、C 表达式、函数名或汇编语言标号。它必须是先前用 ma 命令定义，并有关键字 P/R(input port)或 P/R/W(input/output port)。为 I/O 端口定义的地址范围长度可以是 0x1000～0x1FFF 字节，不必是 16 的倍数。

(2) Page　用来识别此存储器区域的内容。Page＝1，表示该页属于数据存储器。Page=2，表示该页属于 I/O 空间。

(3) Length　定义此空间的范围，此参数可以是任何 C 表达式。

(4) Filename　可以为任何文件名。从连接口或存储器空间读文件时，文件必须存在，否则 mc 命令会失败。

(5) Fileaccess　识别 I/O 和数据存储器的访问特性，必须为表 5-2 所列关键字的一种。

表 5-2　存储器的访问特性对应的关键字

访问文件的类型	访问特性
输入口(I/O 空间)	P|R
输入 EOF，停止软仿真(I/O 端口)	R|P|NR
输出口(I/O 空间)	P|W
内部只读存储器	R
外部只读存储器	EXIR
内部存储器输入 EOF，停止软仿真	R|NR
外部存储器输入 EOF，停止软仿真	EX| R|NR
只写内部存储器空间	W
只写外部存储器空间	EX|W

对于 I/O 存储器空间，当相关的端口地址处有读写指令时，说明有文件访问。任何 I/O 端口都可以同文件相连，一个文件可以同多个端口相连，但一个端口至多与一个输入文件和一个输出文件相连。

如果使用了参数 NR，软仿真读到 EOF 时会停止执行并在命令窗口显示相应信息：

```
<addr>EOF reached — connected at port (I/O_PAGE)
或
<addr>EOF reached — connected at location(DATA_PAGE)
```

此时可以用 mi 命令脱开连接，mc 命令添加新文件。如果未进行任何操作，输入文件会自动从头开始自动执行，直到读出 EOF。如果未定义 NR，则 EOF 被忽略，执行不会停止。输入文件自动重复操作，软件仿真器继续读文件。

例如：设有两个数据存储器块：

```
ma  0x100 ,1,0x10,EX|RAM|   ;block1
ma  0x200,1,0x10,RAM        ;block2
```

可以使用 mc 命令将输入文件连接到 block1：

```
mc  0x100,1,0x1,my_input.dat,EX|R
```

可以使用 mc 命令将输出文件连接到 block2：

```
mc  0x205,1,0x1,my_output.dat,W
```

可以使用 mc 命令，使遇到输入文件的 EOF 时暂停仿真器：

```
mc  0x100,1,0x1,my_input.dat,EX|RNR 或
mc  0x100,1,0x1,my_input.dat,ERNR
```

【例 5.1】 将输入口连接到输入文件。

假定 in.dat 文件中包含的数据是十六进制格式，且一个字写一行，则：

```
0A00
1000
2000
```

使用 ma 和 mc 命令来设置和连接输入口：

```
ma  0x50,2,0x1,R|P          ；将口地址 50H 设置为输入口
mc  0x50,2,0x1,in.dat,R     ；打开文件 in.dat，并将其连接到口 50H
```

假定下列指令是程序中的一部分，则可完成从文件 in.dat 中读取数据：

```
PORTR 0x50,data_mem         ；读取文件 in.dat，并将读取的值放入 data_mem 区域
```

3. 脱开 I/O 端口

使用 md 命令从存储器映射中消去一个端口之前，必须使用 mi 命令脱开该端口。mi(memory disconnect)将一个文件从一个 I/O 端口脱开。其语法如下

```
mi  portaddress，page，{R|W|EX}
```

命令中的端口地址和页是指要关闭的端口，read/write 特性必须与端口连接时的参数一致。

4. 实例

1) 编写汇编语言源程序从文件中读数据

(1) 定义 I/O 端口　使用 ma 命令指定 I/O 端口，在命令窗口输入：

```
ma  0x100,2,0x1,P|R         ；定义地址 0x100 为输入端口
ma  0x102,2,0x1,P|W         ；定义地址 0x102 为输出端口
ma  0x103,2,0x1,P|R|W       ；定义地址 0x103 为输入/输出端口
```

(2) 连接 I/O 端口　用 mc 命令将 I/O 端口连接到输入/输出文件。允许将数据区的任何

区域(除 00H～1FH)连接到输入/输出文件来读写数据。当连接读文件时应确保文件存在。

```
mc  0x100,2,0x1,ioread.txt,R
mc  0x102,2,0x1,iowrite.txt,W
```

为了验证 I/O 端口是否被正确定义，文件是否被正确连接，在命令窗口使用 ml 命令，Simulator 将列出 memory 以及 I/O 端口的配置和所连接的文件名。

(3) 编写汇编语言源程序从文件中读数据：

```
(*ar1+)=port(0x100)；将端口 0x100 所连接文件内容读到 ar1 寄存器指定的地址单元中。
port(0x102)=*ar1    ；将 ar1 寄存器所指地址的内容写到端口 0x102 连接的文件中。
```

2) 脱开 I/O 端口

```
mi 0x100,2,R        ；将 0x100 端口所连接的文件 ioread.txt 从 I/O 端口脱开
mi 0x102,2,W        ；将 0x102 端口所连接的文件 iowrite.txt 从 I/O 端口脱开
```

注意：必须将 I/O 端口脱开，数据才能避免丢失。

5.4　DSP/BIOS 的功能

5.4.1　DSP/BIOS 简介

DSP/BIOS 是一个实时操作系统内核。主要应用在需要实时调度和同步的场合。此外，通过使用虚拟仪表，它还可以实现主机与目标机的信息交换。DSP/BIOS 提供了可抢占线程，具备硬件抽象和实时分析等功能。

DSP/BIOS 由一组可拆卸的组件构成。应用时只需将必需的组建加到工程中即可。DSP/BIOS 配置工具允许通过屏蔽去掉不需要的 DSP/BIOS 特性来优化代码体积和执行速度。

在软件开发阶段，DSP/BIOS 为实时应用提供底层软件，从而简化实时应用的系统软件设计，节约开发时间。更为重要的是，DSP/BIOS 的数据获取(Data Capture)、统计(Statistics)和事件记录功能(Event Logging)在软件调试阶段与主机 CCS 内的分析工具 BIOScope 配合，可以完成对应用程序的实时探测(Probe)、跟踪(Trace)和监控(Monitor)，与 RTDX 技术和 CCS 可视化工具相配合，除了可以直接实时显示原始数据(二维波信号或三维图像)外，还可以对原始数据进行处理，进行数据的实时 FFT 频谱分析、星座图和眼图处理等。

DSP/BIOS 包括如下工具和功能：

(1) DSP/BIOS 配置工具。程序开发者可以利用该工具建立和配置 DSP/BIOS 目标。该工具还可以用来配置存储器、线程优先级和中断处理函数等。

(2) DSP/BIOS 实时分析工具。该工具用来测试程序的实时性。

(3) DSP/BIOS API 函数。应用程序可以调用超过 150 个 DSP/BIOS API 函数。

5.4.2　一个简单的 DSP/BIOS 实例

本节通过一个简单的例子来介绍如何使用 DSP/BIOS 创建、生成、调试和测试程序。该实例就是常用的显示“hello world”程序。在这里没有使用标准 C 输出函数而是使用

DSP/BIOS 功能。利用 CCS2 的剖析特性可以比较标准输入函数和利用 DSP/BIOS 函数执行的性能。值得注意的是，开发 DSP/BIOS 应用程序不仅要有 Simulator(软件调试仿真)，还需要使用 Emulator(硬件仿真)和 DSP/BIOS 插件(安装时装入)。

1. 创建一个配置文件

为使用 DSP/BIOS 的 API 函数，一个程序必须有一个配置文件用来定义程序所需的 DSP/BIOS 对象。

(1) 在 C:\ti\myprojects 目录下新建一个新文件夹 HelloBios。

(2) 将文件夹 C:\ti\tutorial\sim54xx\hello1 下的全部文件复制到新建立的文件夹 HelloBios 中。

(3) 运行 CCS，并打开 C:\ti\myprojects\HelloBios 下的 hello.pjt。

(4) CCS 会弹出如图 5.32 所示的对话框，提示没有找到库文件，这是因为工程被移动了。单击 Browse 按钮，在 C:\ti\c5400\cgtools\lib 找到 rts.lib 库文件。

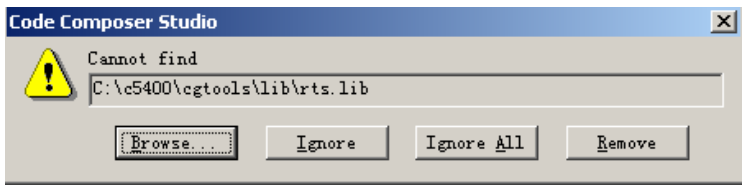

图 5.32　未找到库文件提示框

(5) 单击 hello.pjt、Libraries 和 Source 旁边的“+”号，展开工程视图。

(6) 双击 hello.c 程序，可以看出本程序通过 puts("hello world!\n")函数输出 hello world!。

(7) 编译、下载和运行程序，输出“hello world!”。下面修改程序，使用 DSP/BIOS 输出“hello world!”。

```
#include <stdio.h>
#include "hello.h"
#define BUFSIZE 30
struct PARMS str =
{
    2934,
    9432,
    213,
    9432,
    &str
};
/*
 *  ======== main ========
 */
void main()
{
#ifdef FILEIO
    int     i;
```

```
    char    scanStr[BUFSIZE];
    char    fileStr[BUFSIZE];
    size_t  readSize;
    FILE    *fptr;
#endif
    /* write a string to stdout */
    puts("hello world!\n");
#ifdef FILEIO
    /* clear char arrays */
    for (i = 0; i < BUFSIZE; i++)
    {
    scanStr[i] = 0        /* deliberate syntax error */
       fileStr[i] = 0;
    }
    /* read a string from stdin */
    scanf("%s", scanStr);
    /* open a file on the host and write char array */
    fptr = fopen("file.txt", "w");
    fprintf(fptr, "%s", scanStr);
    fclose(fptr);
    /* open a file on the host and read char array */
    fptr = fopen("file.txt", "r");
    fseek(fptr, 0L, SEEK_SET);
    readSize = fread(fileStr, sizeof(char), BUFSIZE, fptr);
    printf("Read a %d byte char array: %s \n", readSize, fileStr);
    fclose(fptr);
#endif
}
```

(8) 执行菜单命令 File→New→DSP/BIOS Configuration。

(9) 选择与您的 DSP 仿真器相对应的模板并单击 OK 按钮确认。此时将弹出一个新窗口。窗口左半部分为 DSP/BIOS 模块及对象名，右半部分为模块和对象的属性。

(10) 右键单击 LOG-Event Log Manager，在弹出菜单中选择 Insert Log，此时创建一个被称为 LOG0 的 LOG 对象。

(11) 右键单击 LOG0 对象，在弹出菜单中选择 Rename，对象更名为 trace。

(12) 将配置文件存为 hello.cbd，存盘到 C:\ti\myprojects\HelloBios 中，此时将产生以下文件：

① hello.cdb 配置文件，保存配置设置。

② hellocfg.cmd：链接命令文件。

③ hellocfg.s54：汇编语言源文件。

④ hellocfg.h54：myhellocfg.s54 包含的头文件

⑤ hellocfg.h：DSP/BIOS 模块头文件。

⑥ hellocfg_c.c：CSL 结构体和设置代码。

2. 将 DSP/BIOS 添加到工程中

下面将刚才存盘时生成的文件添加到工程文件中。

(1) 执行菜单命令 Project→Add Files to Project，将 hello.cbd 加入，此时工程视图中将添加一个名为 DSP/BIOS Config 的目录，hello.cbd 被列在该目录下。

(2) 链接输出的文件名必须与.cdb 文件名一样，在 Project→ Build Options 的 Linker 栏中将输出文件名修改为 hello.out。

(3) 执行菜单命令 Project→Add Files to Project，将 hellocfg.cmd 加入 CCS 中。由于工程中只能有一个链接命令文件，因此产生如图 5.33 所示的警告信息。

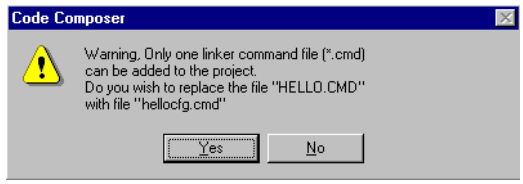

图 5.33　链接命令文件警示

(4) 单击 Yes 按钮，用 hellocfg.cmd 替换原来的 hello.cmd 命令文件。

(5) 在 Project 视图中移去 Vector.asm，这是因为硬件中断矢量已在 DSP/BIOS 配置中自动定义。

(6) 移去 rts.lib 文件，因为此运行支持库也已在 hellocfg.cmd 中指定，链接时将自动加入。

(7) 将 hello.c 文件内容修改为以下代码。LOG_printf 和 put 函数占用相同的资源。

```
#include <std.h>
#include <log.h>
#include "hellocfg.h"
/*
 *  ======== main ========
 */
Void main()
{
   LOG_printf(&trace, "hello world!");
   /* fall into DSP/BIOS idle loop */
   return;
}
```

在以上程序代码中：

① 程序首先包含了 std.h 和 log.h 两个头文件。所有使用 DSP/BIOS API 的程序必须包含 std.h 头文件。此外还应包括该模块使用的头文件，本例中的 LOG 模块头文件为 log.h。在 log.h 中定义了 LOG_Obj 结构，并在 LOG 模块中声明 API 操作。在头文件中，std.h 必须放在其他文件前面，其余模块的先后次序则并不重要。

② 程序中使用关键字 extern，声明在配置文件中创建的 LOG 对象。

③ 主函数调用 LOG_pritf 函数并将 LOG 对象&trace 和 hello world 信息作为参数传给主函数。

④ 主函数返回，程序将进入 DSP/BIOS 等待循环状态，等待软件和硬件中断发生。

(8) 保存 hello.c。

(9) 执行菜单命令 Project→Build Option，直接将 Compiler 栏的命令行参数-d FILEIO 删除。

(10) 重新编译程序。

3. 用 CCS 测试

由于程序只有一行，因此没有必要分析程序。下面对程序进行测试。

(1) 执行菜单命令 File →Load Program，加载 myhello.out。

(2) 执行菜单命令 Debug →Go main，编辑窗口显示 hello.c 文件内容且 main 函数的第一行被高亮显示，表明程序执行到此后暂停。

(3) 执行菜单命令 DSP/BIOS→Message Log，此时将在 CCS 窗口下方出现 Message Log 区域。

(4) 在 Log Name 栏选择 trace 作为要观察的 LOG 名。

(5) 运行程序将在 Message Log 区域出现“hello world！”信息。

(6) 在 Message Log 区域右击并选择 Close，为下面使用剖切(Profiler)作准备。

4. 分析 DSP/BIOS 代码执行时间

下面使用剖切(Profiler)获得 LOG _printf 的执行时间。

(1) 执行菜单命令 File→Reload Program，重新加载程序。

(2) 执行菜单命令 Profiler→Enable Clock，使能时钟

(3) 双击 hello.c，查看源代码。

(4) 执行菜单命令 ViewMax Source/ASM，同时显示 C 及相应汇编代码。

(5) 将光标放在 LOG_printf(&trace, "hello world!");行。

(6) 在 Project 工具栏上的 Toggle Profile Point 图标，设置剖切点。

(7) 将光标移至程序最后一行花括号处，设置第二个剖切点。虽然 return 是程序的最后一条语句，但不能将剖切点放在此行，因为此行不包含等效汇编代码。如果将剖切点放在此行，则 CCS 运行时自动纠正此错误。

(8) 执行菜单命令 Profiler→Start New Session，弹出 Profile Session Name 窗口，取默认名字，单击 OK 按钮，出现 Profile Statistics 窗口。

(9) 运行程序。

(10) 可以看到第二个剖切点的指令周期约为 58，即为执行 LOG_printf 的时间。调用 LOG_printf 比调用 C 中的 puts 函数更为有效，这是因为字符格式串格式是在主机上而不是像 puts 函数那样在目标 DSP 上处理。使用 LOG_printf 函数监视系统状态对程序的实时运行影响比使用 puts 函数小得多。

(11) 停止程序运行。

(12) 执行以下操作以释放被 Profile 任务占用的资源。

① 执行菜单命令 Profiler→Enable Clock，禁止时钟。

② 关闭 Profile Statistics 窗口。

③ 执行菜单命令 Profiler→profile points 删除所有剖切点。

④ 执行菜单命令 View→Mixed Source/ASM，取消 C 与汇编的混合显示。
⑤ 关闭所有源文件和配置窗口。
⑥ 执行菜单命令 Project→Close，关闭工程。

5.5　习题与思考题

1. 简述 CCS 软件配置步骤。
2. CCS 提供了哪些菜单和工具条？
3. 编写一个能显示“This is my program!”的 DSP 程序。
4. 编写程序用 CCS 仿真 INT2 中断。
5. 用 DSP/BIOS 的 LOG 对象方法实现“This is my program!”的输出。