第4章 CCS集成开发环境

4.1 CCS系统安装与设置

4.2 CCS菜单和工具栏

4.3 CCS中的编译器、汇编器和链接器选项设置

4.4 CCS集成开发环境应用

4.5 GEL工具

CCS是Code Composer Studio的缩写，即代码设计工作室。它是TI公司推出的集成可视化DSP软件开发工具。DSP CCS内部集成了以下软件工具：

- DSP代码产生工具（包括DSP的C编译器、汇编优化器、汇编器和链接器）
- CCS集成开发环境（包括编辑、建立和调试DSP目标程序）
- 实时基础软件DSP/BIOS（必须具有硬件开发板）
- RTDX、主机接口和API（必须具有硬件开发板）

在CCS下，用户可以对软件进行编辑、编译、调试、代码性能测试（profile）和项目管理等工作。CCS可以提供如下功能：

- 设置断点
- 在断点处自动修改窗口
- 观察变量
- 观察和编辑存储器和寄存器
- 利用测试点使数据流在目标系统和文件之间流动

- 观察调用堆栈
- 观察图形信号
- 代码性能测试（**profiling**）
- 观察反汇编和C指令执行
- 提供**GEL**（通用扩展语言）语言。此语言能增加一个函数或功能到**CCS**菜单中来完成用户自己设定的任务，是扩展**CCS**功能的专用语言。

4.1 CCS系统安装与设置

4.1.1 CCS系统安装

4.1.2 CCS中DSP开发配置

4.1.1 CCS系统安装

CCS对PC机的最低要求为Windows 95、32M RAM、100M剩余硬盘空间、奔腾90以上处理器、SVGA显示器(分辨率800×600以上)。

进行CCS系统安装时，先将CCS安装盘插入CD-ROM驱动器中，运行光盘根目录下的setup.exe，按照安装向导的提示将CCS安装到硬盘中。安装完成后，安装程序将自动在计算机桌面上创建如图4-1所示的“CCS 2（’C5000）”，“Setup CCS 2（‘C5000）”等快捷图标。

图4-1 “CCS 2（‘C5000）”和“Setup CCS 2 （‘C5000）”快捷图标

4.1.2 CCS中DSP开发配置

在安装CCS之后、运行CCS软件之前，首先需要运行CCS设置程序，根据用户所拥有的软、硬件资源对CCS进行适当的配置。

启动Setup CCS 2（’C5000）应用程序，单击Close按钮关闭Import Configuration对话框，将显示Code Composer Studio Setup窗口，如图4-2所示。

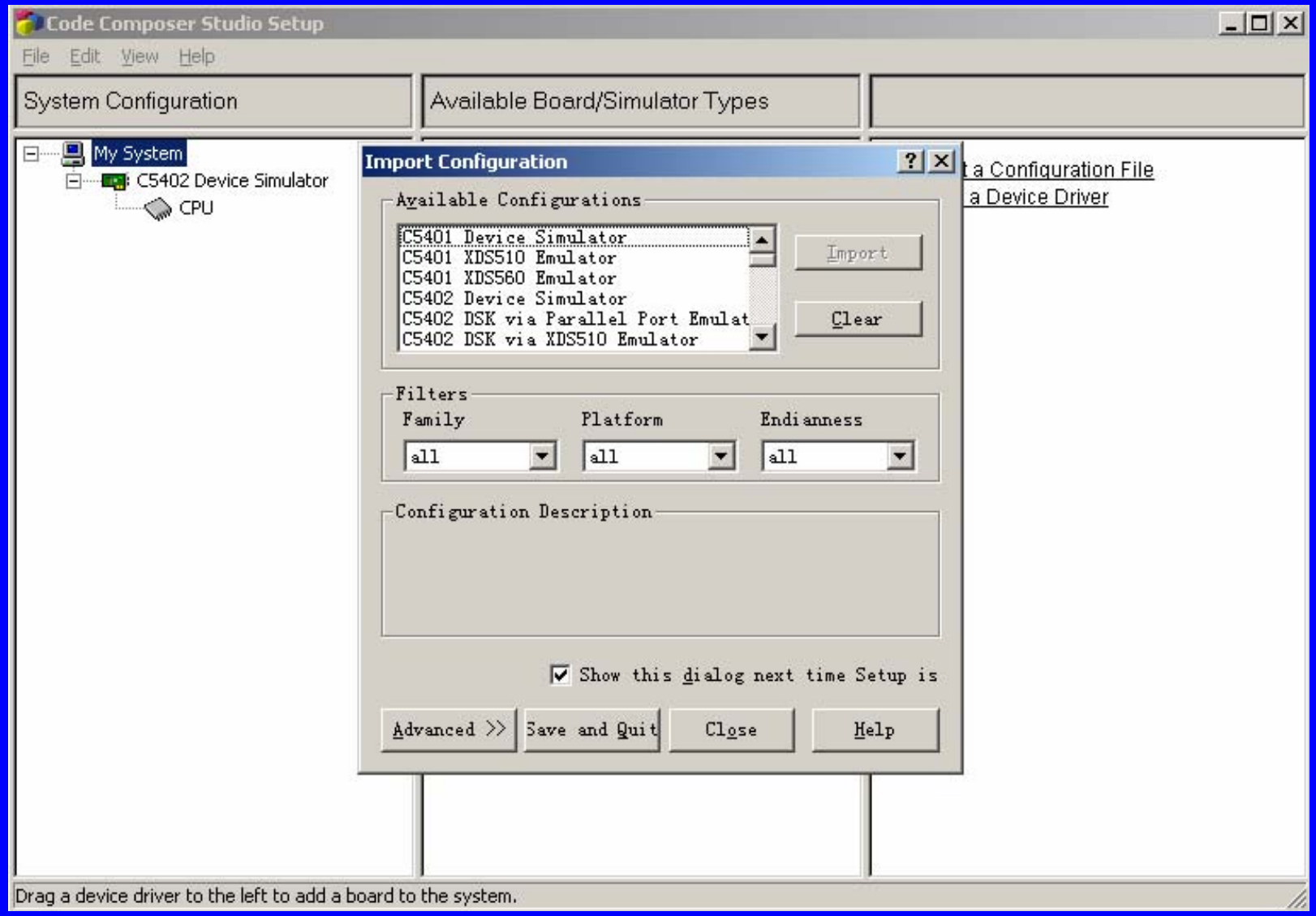


图4-2 Code Compuser studio Setup窗口

4.2 CCS菜单和工具栏

4.2.1 菜单

4.2.2 工具栏

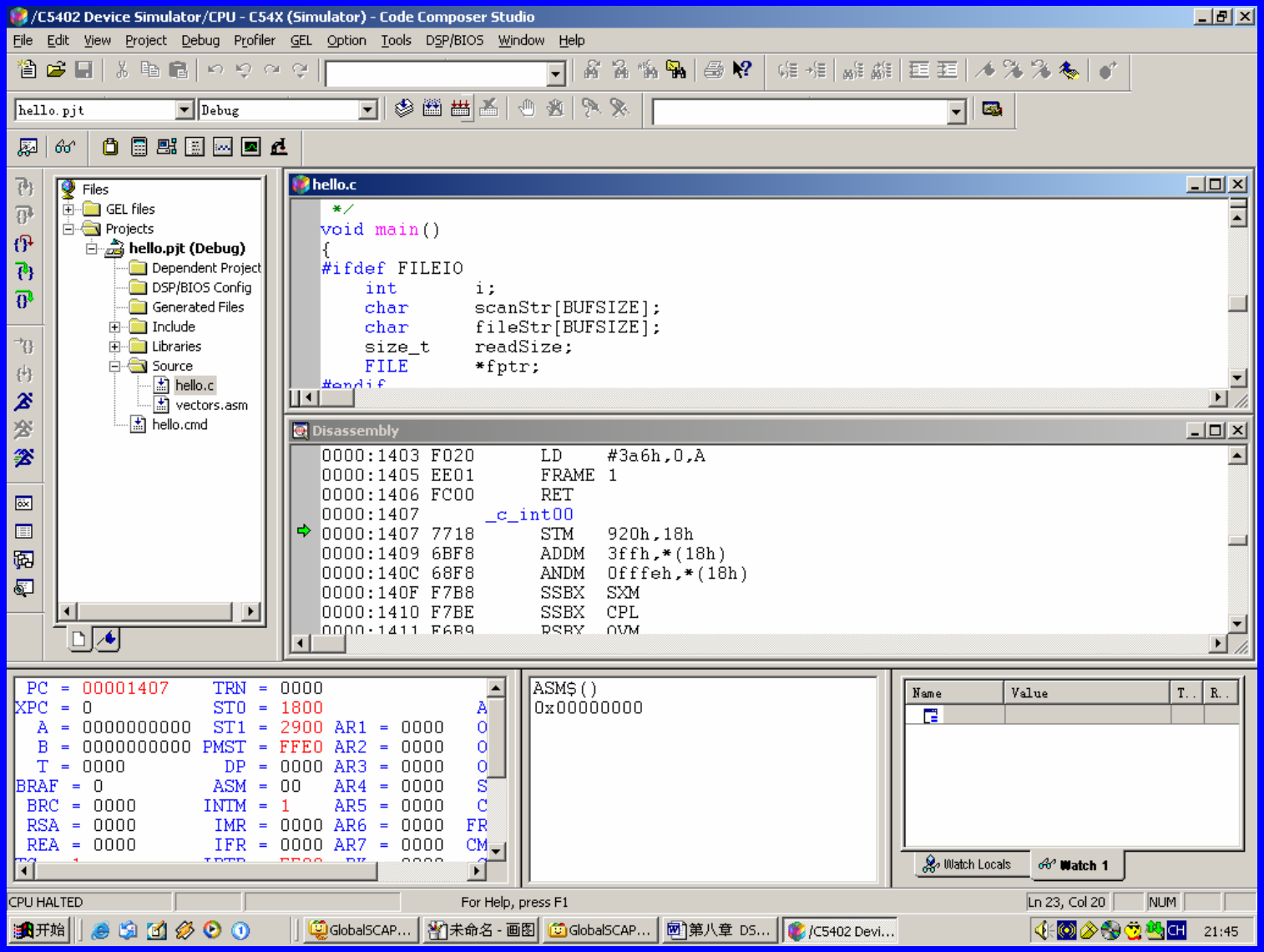


图4-3 CCS运行主窗口1

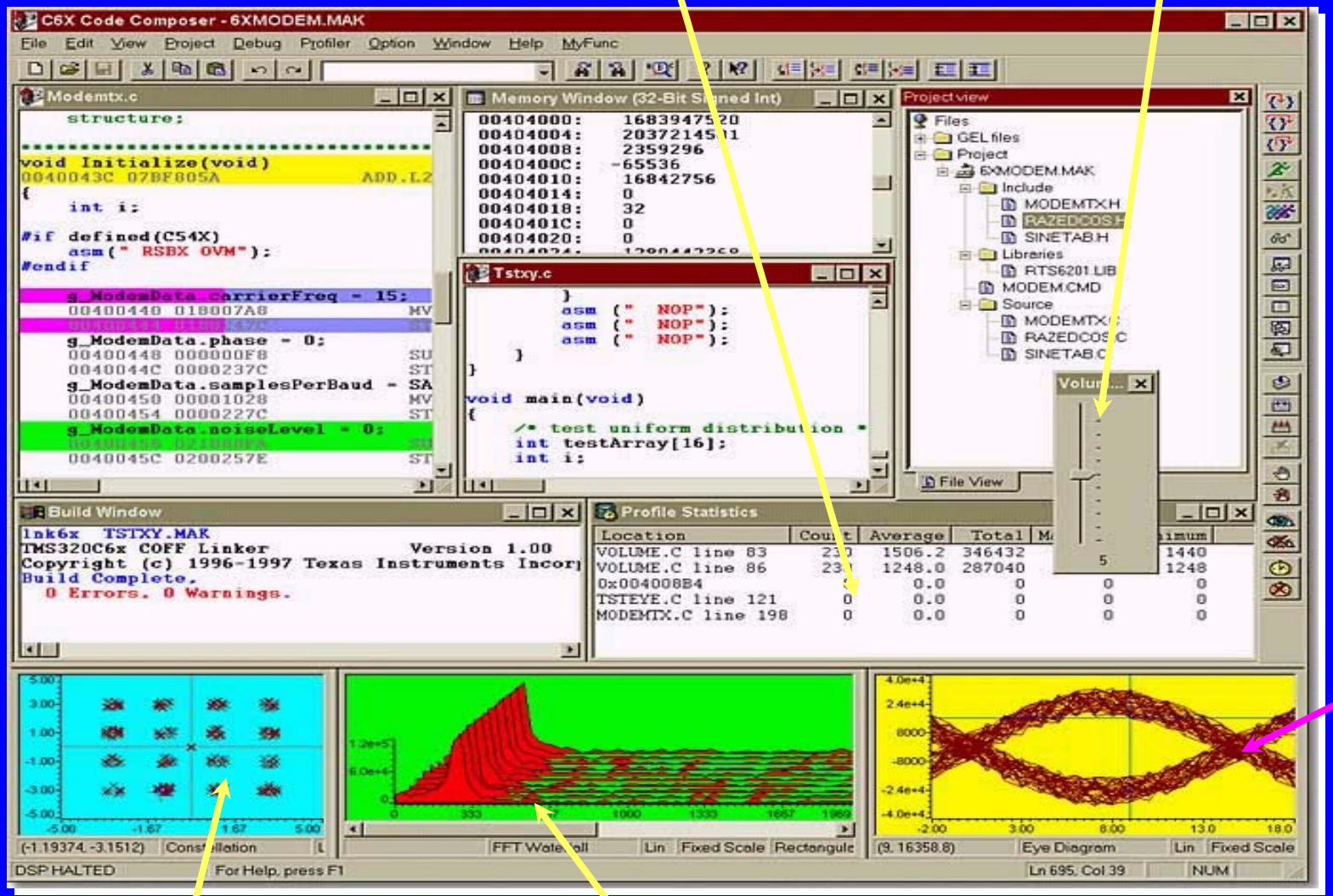


图4-4 CCS运行主窗口2

4.2.1 菜单

1. File菜单

表4-1 File菜单

菜单命令		功能
New	Source File	新建一个源文（.c，.asm，.h，.cmd，.gel，.map，.inc 等）
	DSP/BIOS Config	新建一个 DSP/BIOS 配置文件
	Visual Linker Recipe	打开一个 Visual Linker Recipe 向导
Load Program		将 COFF（.out）文件中的数据和符号加载到目标板（实际目标板或 Simulator）
Reload Program		重新加载 COFF 文件，如果程序未作更改则只加载程序代码而不加载符号表
Data	Load	将 PC 文件中的数据加载到目标板，可以指定存放的地址和数据长度，数据文件可以是 COFF 文件格式，也可以是 CCS 支持的数据格式
	Save	将目标板存储器数据存储到一个 PC 数据文件中
Workspace	LoadWorkspace	装入工作空间
	Save Workspace	保存当前的工作环境，即工作空间，如父窗、子窗、断点、探测点、文件输入/输出、当前的工程等。
	Save WorkspacAs	用另外一个不同的名字保存工作空间
File I/O		CCS 允许在 PC 文件和目标 DSP 之间传送数据。File I/O 功能应与 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，实时数据交换应使用 RTDX

CCS中DATA→LOAD/SAVE的数据文件格式

该数据文件格式为字符格式文件，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1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

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。

CCS中DATA View→Memory 数据文件

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。

2. Edit菜单

表4-2 Edit菜单

菜单命令		功能
Find in Files		在多个文本文件中查找特定的字符串或表达式
Go To		快速跳转到源文件中某一指定行或书签处
Memory	Edit	编辑某一存储单元
	Copy	将某一存储块（标明起始地址和长度）数据复制到另一存储块
	Fill	将某一存储块填入某一固定值
	Patch Asm	在不修改源文件的情况下修改目标 DSP 的执行代码
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
Variable		修改某一变量值。如果目标 DSP 由多个页面构成，则可使用@prog、@data 和@io 分别指定页面是程序区、数据区和 I/O 空间，例如：*0x1000@prog = 0
Command Line		可以方便地输入表达式或执行 GEL 函数
Column Editing		选择某一矩形区域内的文本进行列编辑（剪切、复制及粘贴等）
Bookmarks		在源文件中定义一个或多个书签便于快速定位。书签保存在 CCS 的工作区（Workspace）内以便随时被查找到

3. View菜单

表4-3 View菜单

菜单命令		功能
Dis-Assembly		当将程序加载入目标板后，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调试所需的符号信息。
Memory		显示指定存储器的内容
CPU	CPU Register	显示 DSP 的寄存器内容
Registers	Peripheral Regs	显示外设寄存器内容。Simulator 不支持此功能
Graph	Time/Frequency（时间/频率图形）	在时域或频域显示信号波形。频域分析时将对数据进行 FFT 变换，时域分析时数据无须进行预处理。显示缓冲的大小由 Display Data Size 定义
	Constellation（星座图形）	使用星座图显示信号波形。输入信号被分解为 X、Y 两个分量，采用笛卡尔坐标显示波形。显示缓冲的大小由 Constellation Points 定义
	Eye Diagram（眼图）	使用眼图来量化信号失真度。在指定的显示范围内，输入信号被连续叠加并显示为眼睛的形状
	Image（图像）	使用 Image 图来测试图像处理算法。图像数据基于 RGB 和 YUV 数据流显示
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，还可显示数组、结构或指针等包含多个元素的变量
Project		CCS 启动后将自动打开工程视图。在工程视图中，文件按其性质分为源文件、头文件、库文件及命令文件
Mixed Source/Asm		同时显示 C 代码及相关的反汇编代码（位于 C 代码下方）

4. Project菜单

表4-4 Project菜单

菜单命令	功能
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
Compile File	对 C 或汇编源文件进行编译
Build	重新编译和链接。对于那些没有修改的源文件，CCS 将不重新编译
Rebuild All	对工程中所有文件重新编译并链接生成输出文件
Stop Build	停止正在 Build 的进程
Show Dependencies Scan All Dependencies	为了判别哪些文件应重新编译，CCS 在 Build 一个程序时会生成一棵关系树(Dependency Tree)以判别工程中各文件的依赖关系。使用这两个菜单命令则可以观察工程的关系树
Build Options	用来设定编译器、汇编器和链接器的参数
Recent Project Files	加载最近打开的工程文件

5．Debug菜单

表4-5 Debug菜单

菜单命令	功能
Breakpoints	断点。程序在执行到断点时将停止运行。
Step Into	单步运行。如果运行到调用函数处将跳入函数单步执行
Step Over	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。
Step Out	如果程序运行在一个子程序中，执行 Step Out 将使程序执行完该子程序后回到调用该函数的地方。
Run	从当前程序计数器（PC）执行程序，碰到断点时程序暂停执行
Halt	中止程序运行
Animate	运行程序。碰到断点时程序暂停运行，更新未与任何 Probe Point 相关联的窗口后程序继续运行。
Run Free	忽略所有断点（包括 Probe Point 和 Profile Point），从当前 PC 处开始执行程序。此命令在 Simulator 下无效。
Run to Cursor	执行到光标处，光标所在行必须为有效代码行
Multiple Operation	设置单步执行的次数
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
Restart	将 PC 值恢复到程序的入口。此命令并不开始程序的执行
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

表4-6 Profiler菜单

菜单命令	功能
Start New Session	开始一个新的代码段分析，打开代码分析统计观察窗口
Enable Clock	为了获得指令周期及其他事件的统计数据，必须使能代码分析时钟。代码分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 对话框内修改其值。CLK 还可在用户定义的 GEL 函数中使用。指令周期的计算方式与使用的 DSP 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，其他的驱动程序则可能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分析数据。当时钟使能时，CCS 调试器将占用必要的资源实现指令周期的计数。加载程序并开始一个新的代码段分析后，代码分析时钟自动使能
View Clock	打开 Clock 窗口，显示 CLK 变量的值。双击 Clock 窗口的内容可直接将 CLK 变量复位
Clock Setup	设置时钟。在 Clock Setup 对话框中（如图 6-5 所示），Instruction Cycle Time 域用于输入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换为时间或频率。在 Count 域选择分析的事件。对某些驱动程序而言，CPU Cycles 可能是惟一的选项。对于使用片内分析功能的驱动程序而言，可以分析其他事件，如中断次数、子程序或中断返回次数、分支数及子程序调用次数等。可使用 Reset Option 参数决定如何计数。如选择 Manual 选项，则 CLK 变量将不断累加指令周期数；如选择 Auto 选项，则在每次 DSP 运行前将自动将 CLK 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。

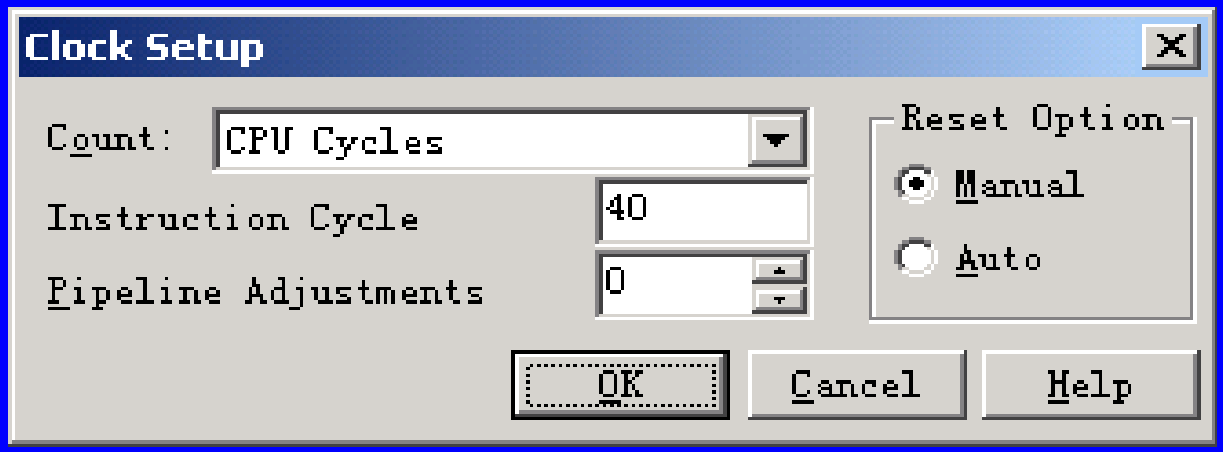

Clock Setup
Count:
CPU Cycles
Instruction Cycle
40
Pipeline Adjustments
0
Reset Option
Manual
Auto
OK
Cancel
Help

图4-5 时钟设置

7. Option菜单

表4-7 Option菜单

菜单命令	功能
Font	设置集成开发环境字体格式及字号大小
Memory Map	用来定义存储器映射，弹出 Memory Map 对话框，如图 6-6 所示。存储器映射指明了 CCS 调试器能访问哪段存储器，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。在对话框中选中 Enable Memory Mapping 以使能存储器映射。第一次运行 CCS 时，存储器映射即呈禁用状态（未选中 Enable Memory Mapping），也就是说，CCS 调试器可存取目标板上所有可寻址的存储器（RAM）。当使能存储器映射后，CCS 调试器将根据存储器映射设置检查其可以访问的存储器。如果要存取的是未定义数据或保护区数据，则调试器将显示默认值（通常为 0），而不是存取目标板上数据。也可在 Protected 域输入另外一个值，如 0XDEAD，这样当试图读取一个非法存储地址时将清楚地给予提示
Disassembly Style	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
Customize	打开用户自定义界面对话窗

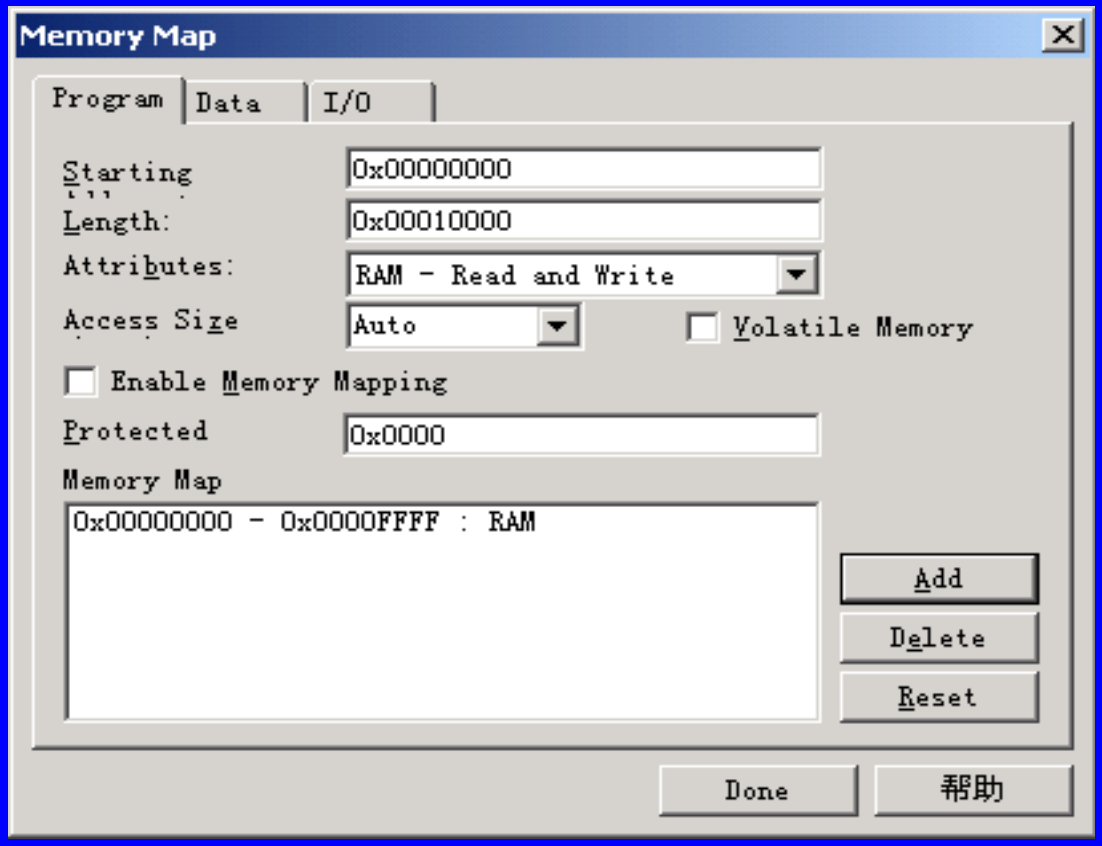


图4-6 Memory Map对话框

8. Tools菜单

表4-8 Tools菜单

菜单命令	功能
Data Converter Support	使开发者能快速配置与 DSP 芯片相连的数据转换器
C54xx McBSP	使开发者能观察和编辑多信道缓冲串行口（McBSP）的内容
C54xx Emulator Analysis	使开发者能设置、监视事件和硬件断点的发生
C54xx DMA	使开发者能观察和编辑 DMA 寄存器的内容
C54xx Simulator Analysis	使开发者能设置和监视事件的发生
Command Window	在 CCS 调试器中键入所需的命令，键入的命令遵循 TI 调试器命令语法格式。例如，在命令窗口中键入 HELP 并回车，可得到命令窗口支持的调试命令列表
Port Connect	将 PC 文件与存储器（端口）地址相连，从而可从文件中读取数据或将存储器（端口）数据写入文件中
Pin Connect	用于指定外部中断发生的间隔时间，从而使用 Simulator 来仿真和模拟外部中断信号：①创建一个数据文件以指定中断间隔时间（用 CPU 时钟周期的函数来表示）；②从 Tools 菜单下选择 Pin Connect 命令；③按 Connect 按钮，选择创建好的数据文件，将其连接到所需的外部中断引脚；④加载并运行程序
Linker Configuration	选择一个工程所用的链接器
RTDX	实时数据交换功能，使开发者在不影响程序执行的情况下分析 DSP 程序的执行情况

4.2.2 工具栏

1. Standard Toolbar

Standard工具栏包括以下常用工具如图4-7所示：

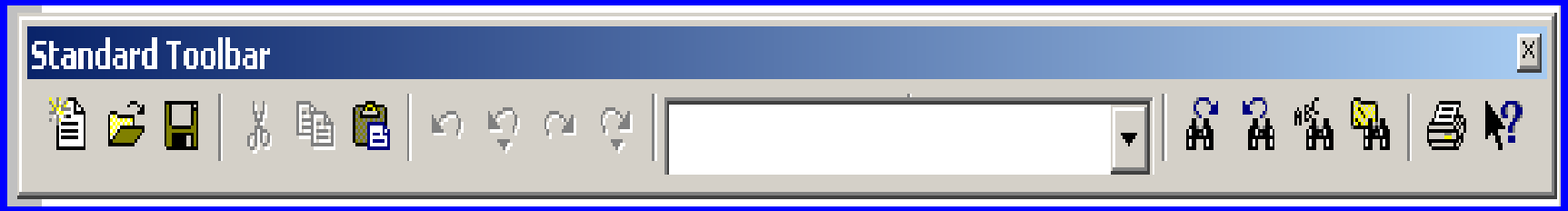


图4-7 Standard工具栏

2．GEL Toolbar

GEL工具栏提供了执行GEL函数的一种快捷方法，如图4-8所示。在工具栏的左侧文本输入框中键入GEL函数名，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。

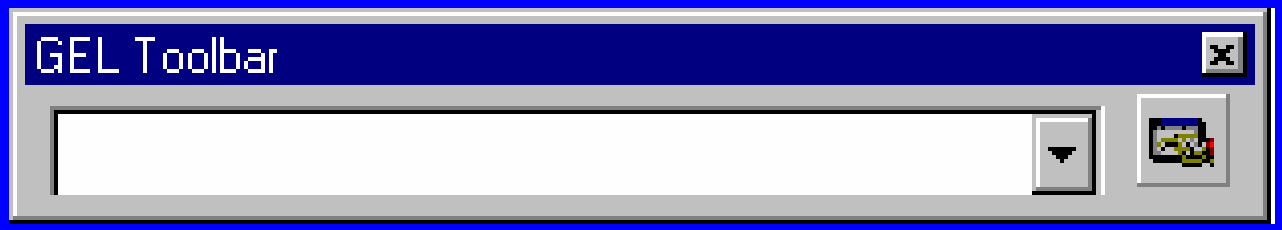


图4-8 GEL工具栏

3．Project Toolbar

Project工具栏提供了与工程和断点设置有关的命令，**Project**工具栏提供了以下命令如图**4-9**所示。

图**4-9 Project**工具栏

4. Debug Toolbar

Debug工具栏提供以下常用的调试命令如图**4-10**所示。

图**4-10** **Debug**工具栏

5. Edit Toolbar

Edit工具栏提供了一些常用的编辑命令及书签命令如图**4-11**所示。

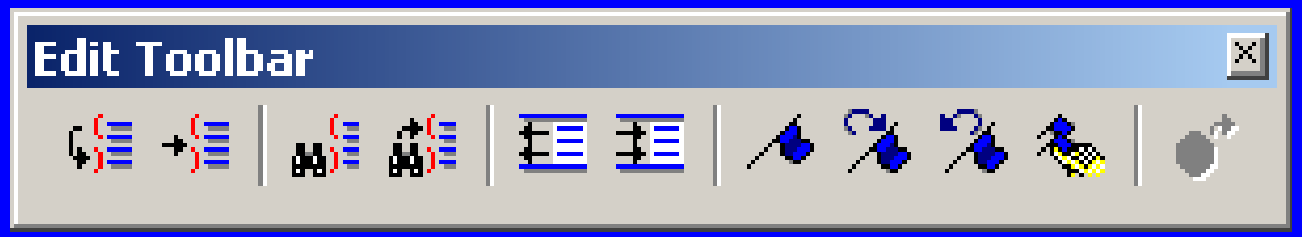


图**4-11 Edit**工具栏

6. Plug-in Toolbars

Plug-in Toolbars包括Watch Window和DSP/BIOS两个窗口，其中Watch Window如图4-12所示。

图4-12 Watch Window 工具栏

4.3 CCS中的编译器、汇编器和链接器选项设置

4.3.1 编译器、汇编器选项

4.3.2 链接器选项

4.3.1 编译器、汇编器选项

编译器（**Compiler**）包括分析器、优化器和代码产生器，它接收**C/C++**源代码并产生**TMS320C54x**汇编语言源代码。

汇编器（**Assembler**）的作用就是将汇编语言源程序转换成机器语言目标文件，这些目标文件都是公共目标文件格式（**COFF**）。如图**4-13**、表**4-9**所示。

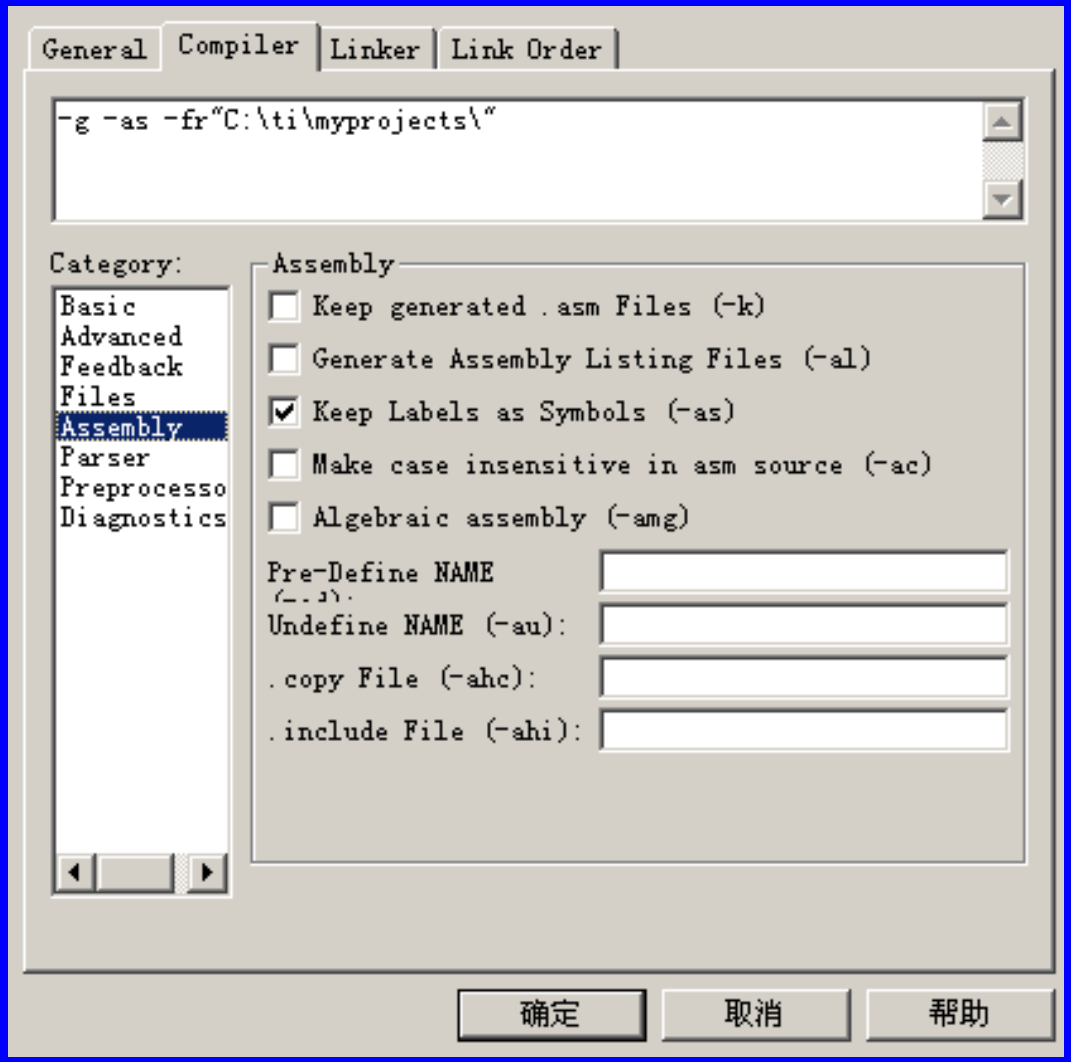


图4-13 生成选项窗口——编译器标签

表4-9 编译器、汇编器常用选项（在Compiler中）

类	域	选项	含义
Basic	Generate Debug Inf	-g	产生由 C/C++源代码级调试器使用的符号调试伪指令，并允许汇编器中的汇编源代码调试
		-gw	产生由 C/C++源代码级调试器使用的 DWARF 符号调试伪指令，并允许汇编器中的汇编源代码调试
Basic	Opt Level（使用 C 优化器）	-o0	控制流图优化，把变量分配到寄存器，安排循环，去掉死循环，简化表达式
		-o1	包括-o0 优化，并可去掉局部未用赋值
		-o2	包括-o1 优化，并可循环优化，去掉冗余赋值，将循环中的数值下标转换成增量指针形式，打开循环体（循环次数很少时）
Advanced	RTS Modifications（结合-o3 选项）	-oL2	取消声明或改变库函数
		-oL1	声明一个标准的库函数
	Auto Inlining Threshold	-oi	设置自动插入函数长度的极限值（仅对-o3 选项）
		-ma	指示所使用的别名技术
		-mr	禁用不可中断的 RPT 指令

类	域	选项	含义
Feedback	Opt Info File（创建优化信息文件）	-on0	不产生优化器的信息文件
		-on1	产生优化器的信息文件
		-on2	产生详细的优化器的信息文件
	Generate Optimizer Comments	-os	将优化器的注释和汇编源文件语句交织在一起
Files	Asm File Extension	-ea	为汇编语言源文件设置新的默认扩展名
	Obj File Extension	-eo	为目标文件设置新的默认扩展名
	Asm Directory	-fs	指定汇编源文件目录
	Obj Directory	-fr	指定目标文件目录
	Temporary File Dir	-ft	指定临时文件目录
	Absolute Listing File Dir	-fb	指定绝对列表文件目录
	Listing/Xref Dir	-ff	指定汇编清单文件和交叉引用列表文件目录

4.3.2 链接器选项

在汇编程序生成代码中，链接器的作用如下：

（1）根据链接命令文件（.cmd文件）将一个或多个COFF目文件链接起来，生成存储器映象文件（.map）和可执行的输出文件（.out文件）。

（2）将段定位于实际系统的存储器中，给段、符号指定实际地址。

（3）解决输入文件之间未定义的外部符号引用（如图4-14、表4-10所示）。

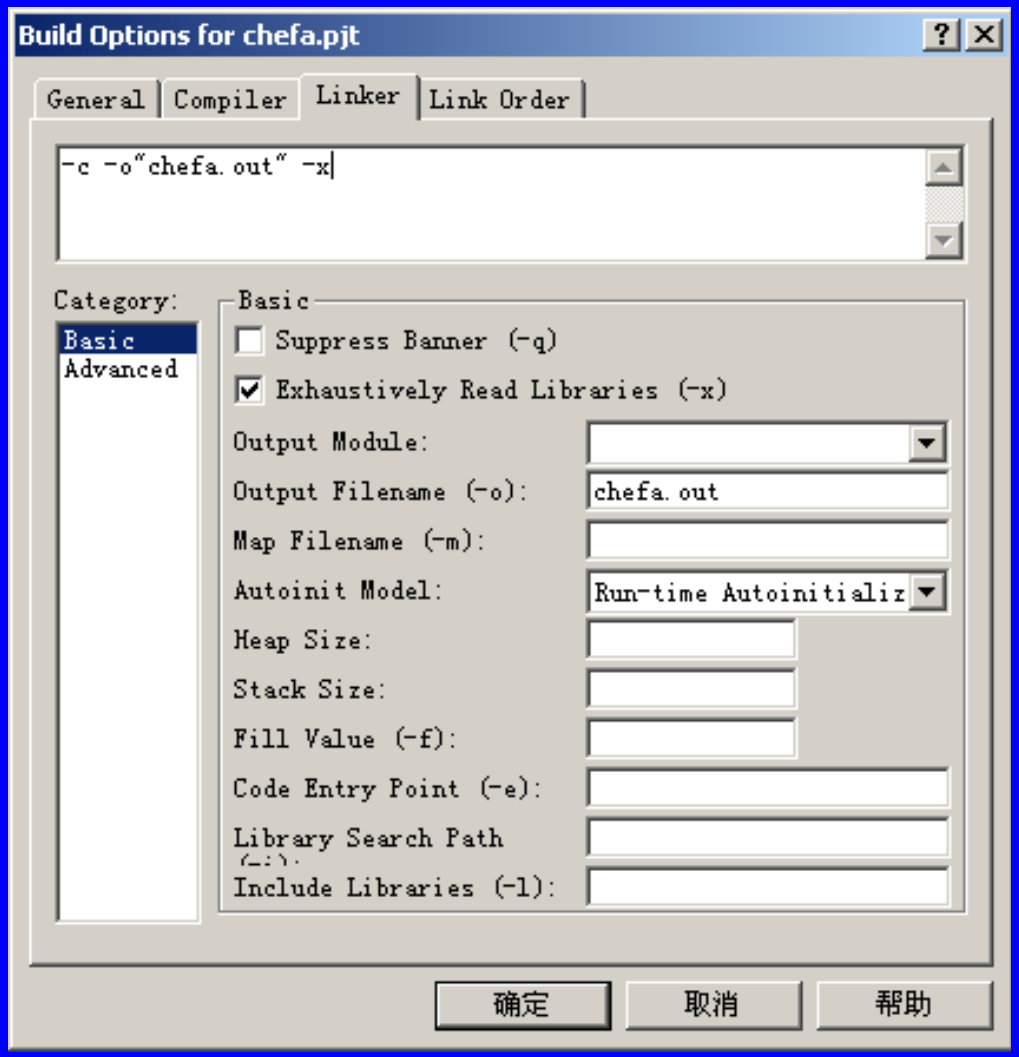


图4-14 生成选项窗口——链接器标签

表4-10 链接器常用选项（在Linker中）

选项	含义
Exhaustively Read Libraries (-x)	迫使重读库，以分辨后面的引用。如果后面引用的符号定义在前面已读过的存档库中，该引用不能被分辨出，采用-x 选项，可以迫使链接器重读所有库，直到没有更多的引用能够被分辨为止
-q	请求静态运行（quiet run），即压缩旗标（banner），必须是在命令行的第一个选项
-a	生成一个绝对地址、可执行的输出模块。所建立的绝对地址输出文件中不包含重新定位信息。如果既不用-a 选项，也不用-r 选项，链接器就像规定-a 选项那样处理
-r	生成一个可重新定位的输出模块，不可执行
-ar	生成一个可重新定位、可执行的目标模块。与-a 选项相比，-ar 选项还在输出文件中保留有重新定位的信息
Map Filename (-m)	生成一个.map 映象文件，filename 是映象文件的文件名。.map 文件中说明了存储器配置、输入、输出段布局以及外部符号重定位之后的地址等
Output Filename (-o)	对可执行输出模块命名，如果缺省，则此文件名为 a.out
-c	C 语言选项用于初始化静态变量，告诉链接器使用 ROM 自动初始化模型

Include Libraries (-l)	命名一个文档库文件作为链接器的输入文件，filename 为文档库的某个文件名。此选项必须出现在-i 选项之后
Stack Size	设置 C 系统堆栈，大小以字为单位，并定义指定堆栈大小的全局符号。默认的 size 值为 1K 字
Heap Size	为 C 语言的动态存储器分配设置堆栈大小，以字为单位，并定义指定的堆栈大小的全局符号，size 的默认值为 1 K 字
Disable Conditional Linking (-j)	不允许条件链接
Disable Debug Symbol Merge (-b)	禁止符号调试信息的合并。链接器将不合并任何由于多个文件而可能存在的重复符号表项，此项选择的效果是使链接器运行较快，但其代价是输出的 COFF 文件较大。默认情况下，链接器将删除符号调试信息的重复条目
Strip Symbolic Information (-s)	从输出模块中去掉符号表信息和行号
Make Global Symbols Static (-h)	使所有的全局符号成为静态变量
Warn About Output Sections (-w)	当出现没有定义的输出段时，发出警告
Define Global Symbol (-g)	保持指定的 global_symbol 为全局符号，而不管是否使用了-h 选项
Create Unresolved Ext Symbol (-u)	将不能分辨的外部符号放入输出模块的符号表

4.4 CCS集成开发环境应用

4.4.1 创建工程（project）文件

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环节。

由于CCS是以工程文件方式进行管理的，所以要在CCS中编译、汇编、链接C程序及汇编程序，首先要创建一个工程文件，然后再将相关程序放入到工程文件中。

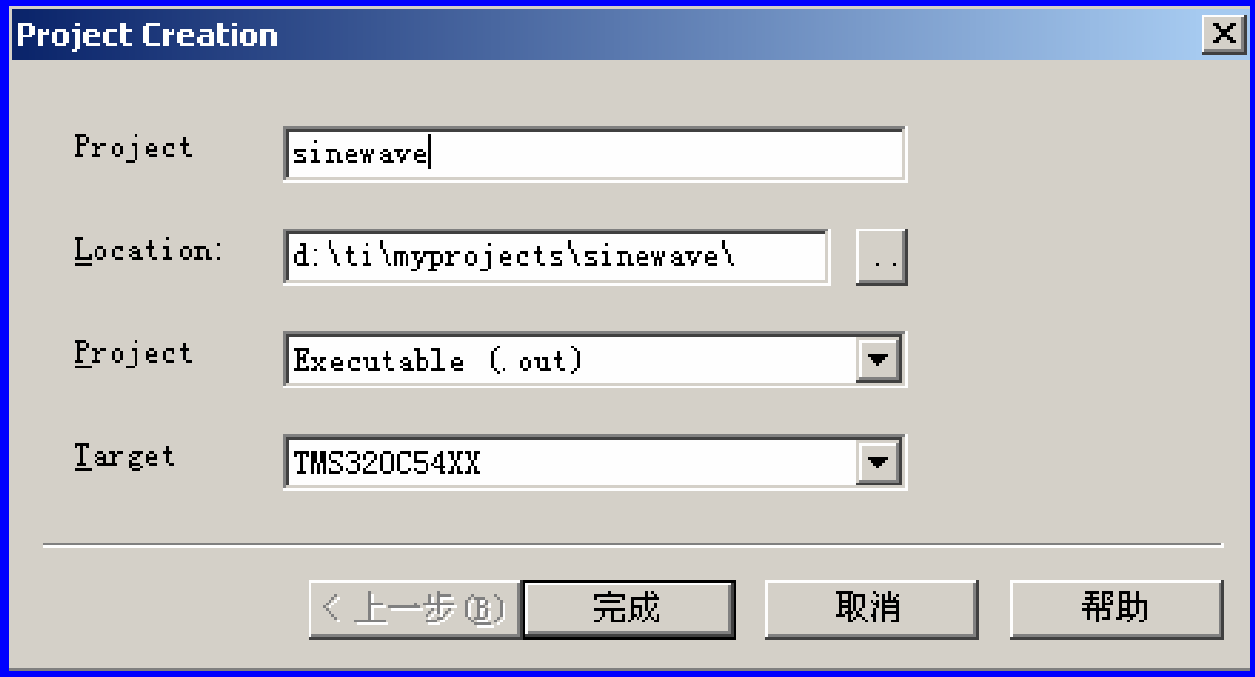


图4-15 创建工程项目

4.4.1.1 向工程中添加文件

首先将安装目录下（**d:\ti\tutorial\sim54xx\sinewave**）的文件**sine.h**，**sine.c**，**sinewave.cmd**拷贝到我们的工程目录下（**d:\ti\myprojects\sinewave**），然后选择**Project→Add Files to Project**，在**d:\ti\myprojects\sinewave**目录中查找**sine.c**，同时打开它。此时系统将**sine.c**文件自动添加到**Project→Source**中。用同样的方法将**sinewave.cmd**文件添加到对应的目录中。（见图**4 - 16**）

注意：***.h**文件不能用上述方法添加到工程，而是利用**Project→Scan All File Dependencies**，系统自动将***.h**文件添加到**include**目录中。

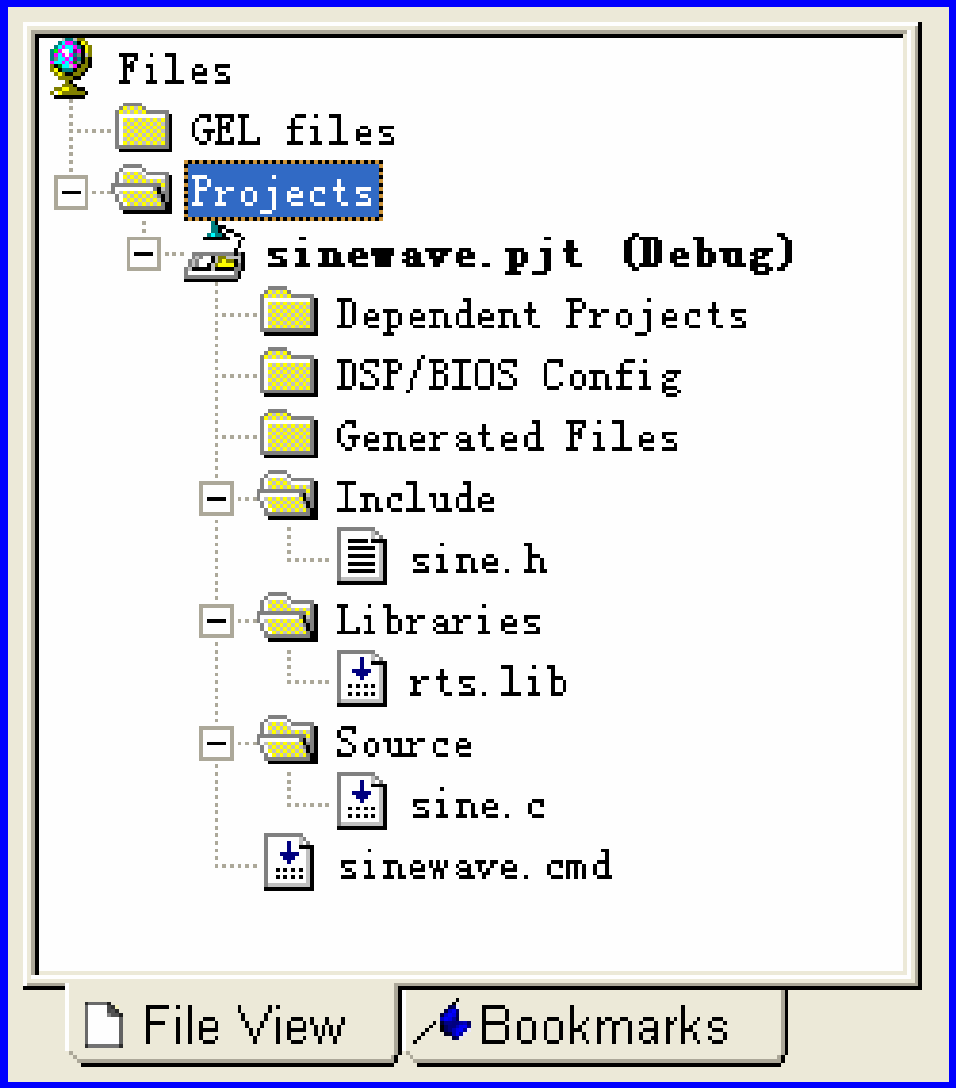

Files
GEL files
Projects
sinewave.pjt (Debug)
Dependent Projects
DSP/BIOS Config
Generated Files
Include
sine.h
Libraries
rts.lib
Source
sine.c
sinewave.cmd
File View
Bookmarks

图4-16 工程视窗

4.4.1.2 编辑文件

通过双击**project→Source**目录下的**sine.c**文件可以查看或编辑源程序，下面我们对**sine.c**文件做如下修改：在语句行**puts(“SineWave example started.\n”);**后加入以下一段程序：

```
for(i=0;i<360;i++)
    a[i]=0;
for(i=0;i<360;i++)
    a[i]=(int)(sin(i*3.14159/180)*32767);
```

4.4.1.3 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，为调试做准备。构建工程的方法如下：

选择**Project→Build**后，系统提示在编译**sine.c**文件时出现两处错误：标志符**i**和**a**未定义。在**sine.c**文件中定义全局变量：**int a[360],i;**并增加语句**#include <math.h>**，然后保存，再次选择**Project→Build**，该错误消除，但系统仍然提示有出错信息。向工程中添加**CCS C**编译器的实时运行支持库文件**rts.lib**后，问题解决。此时，系统自动生成一个可执行文件，**sinewave.out**文件。

注：也可单独完成编译、汇编、链接。

4.4.2 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：载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。

4.4.2 调试

4.4.2.1 载入可执行程序

4.4.2.2 运行程序

4.4.2.3 设置断点

4.4.2.4 观察数据和图形

4.4.2.5 动态图形显示

4.4.2.6 探点的设置及从PC机文件中读取数据

4.4.2.7 代码执行时间分析（Profiler的使用）

4.4.2.1 载入可执行程序

选择**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。在 **d:\ti\myprojects\sinewave\debug** 目录中找 **sinewave.out**文件，同时打开它，这时在反汇编窗口显示该文件的反汇编程序。

4.4.2.2 运行程序

程序运行，可以选择**Debug→Step**单步运行或选择**Debug→Run**全程运行或选择相关图标，可以通过查看内存表等方法，看到程序运行的结果。

4.4.2.3 设置断点

设置断点是最常用的程序调试方法之一。程序在断点处暂停运行，用户可以查看程序状态、检查或修改存储器、查看调用堆栈等。断点主要分为软件断点和硬件断点两类，二者的区别在于，硬件断点不修改目标程序。

设置软件断点可以先把光标放到想设置断点的语句上，然后单击鼠标右键，选择**“Toggle breakpoint”**或者直接单击工具栏中的手掌图案的图标或者双击鼠标左键。

4.4.2.4 观察数据和图形

在程序运行期间，可以在屏幕上跟踪程序的运行结果，帮助你调试程序；可选用数据和图形的方法观察结果。

1.观察数据

（1）使用watch窗口观察数据

①选择View→Watch Window，在CCS窗口的右下角出现watch窗口，程序运行时，这个区显示watch变量的值。

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。例如：在调试sine.c程序时，键入a数组。

在watch窗口列出+a=0x0000081，（单击“+”号，可查看此项的列表，单击“—”号，可折叠展开的列表），0x0000081为数组a的起始地址。当程序运行完之后，点击watch窗口的+a，可显示如下结果：

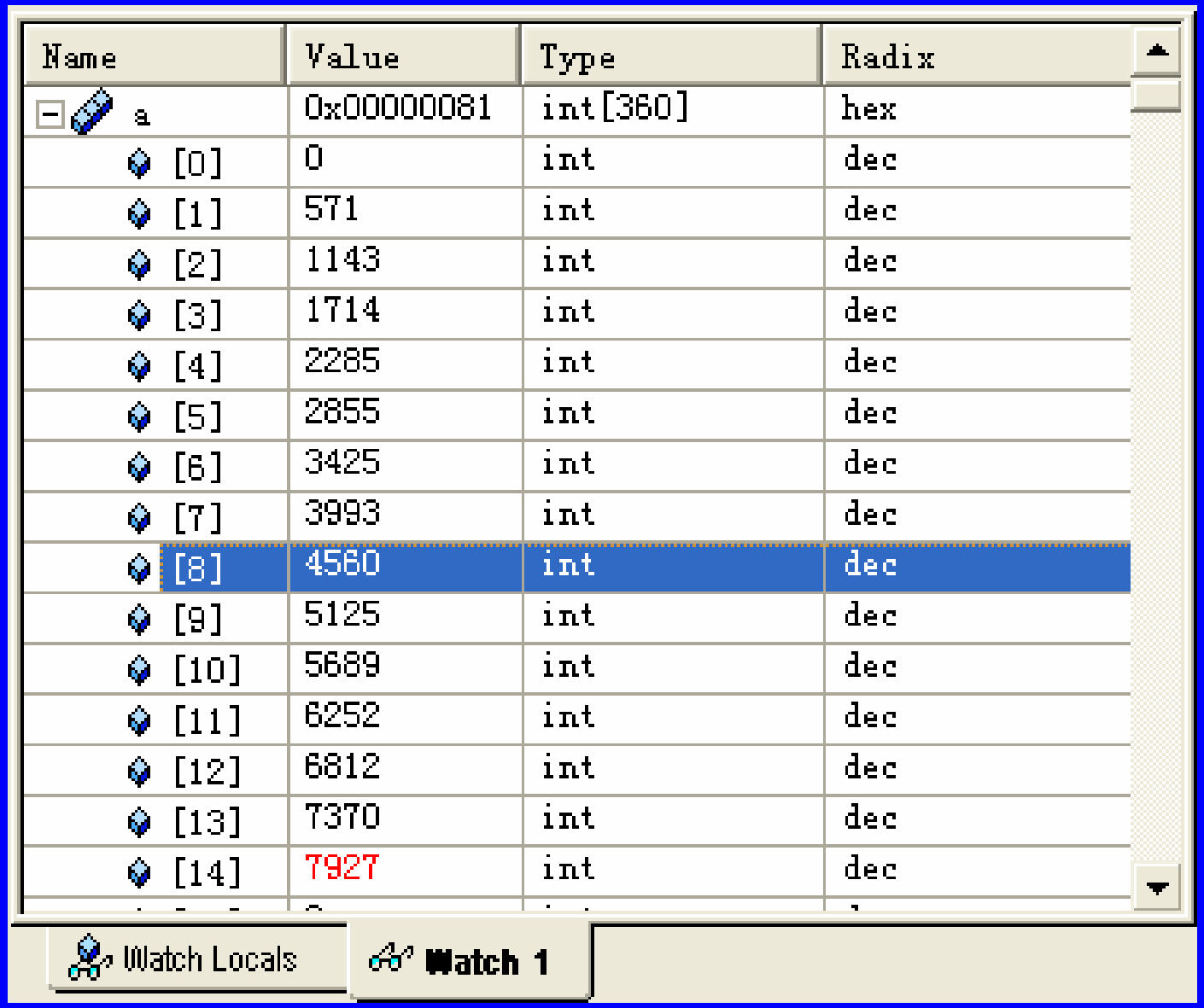


图4-17 使用watch窗口观察数据

（2）使用**memory**窗口观察数据

选择**View→Memory**，出现如图**4-18**所示的提示框。

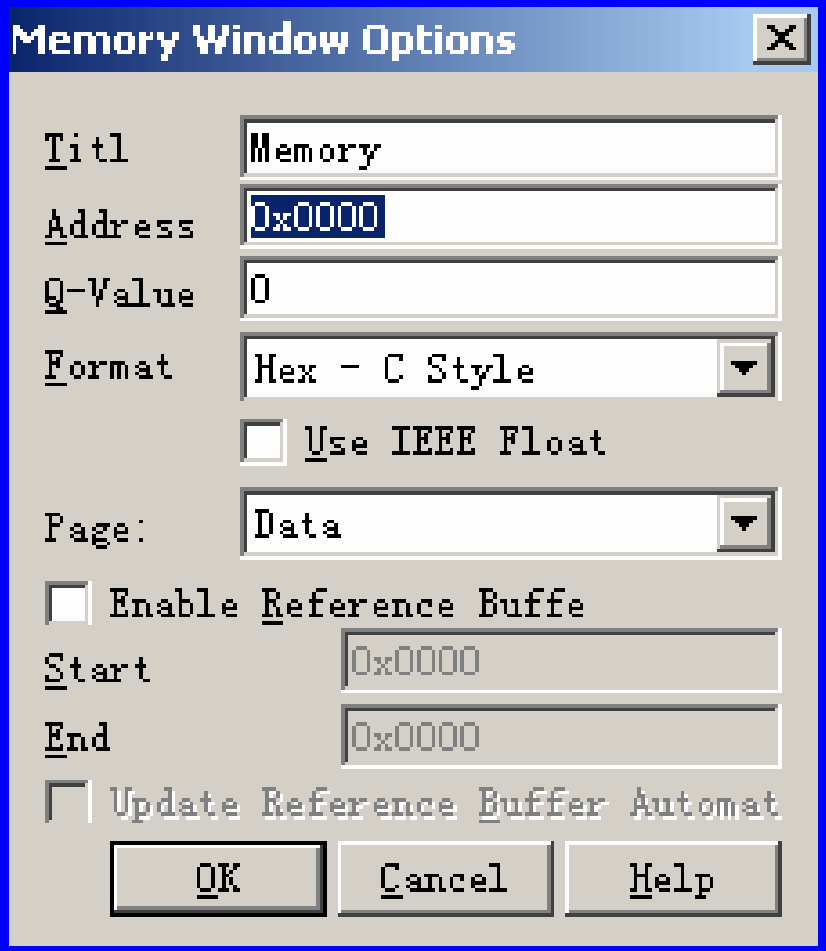


图4-18 存储器窗口观察选项设置对话框

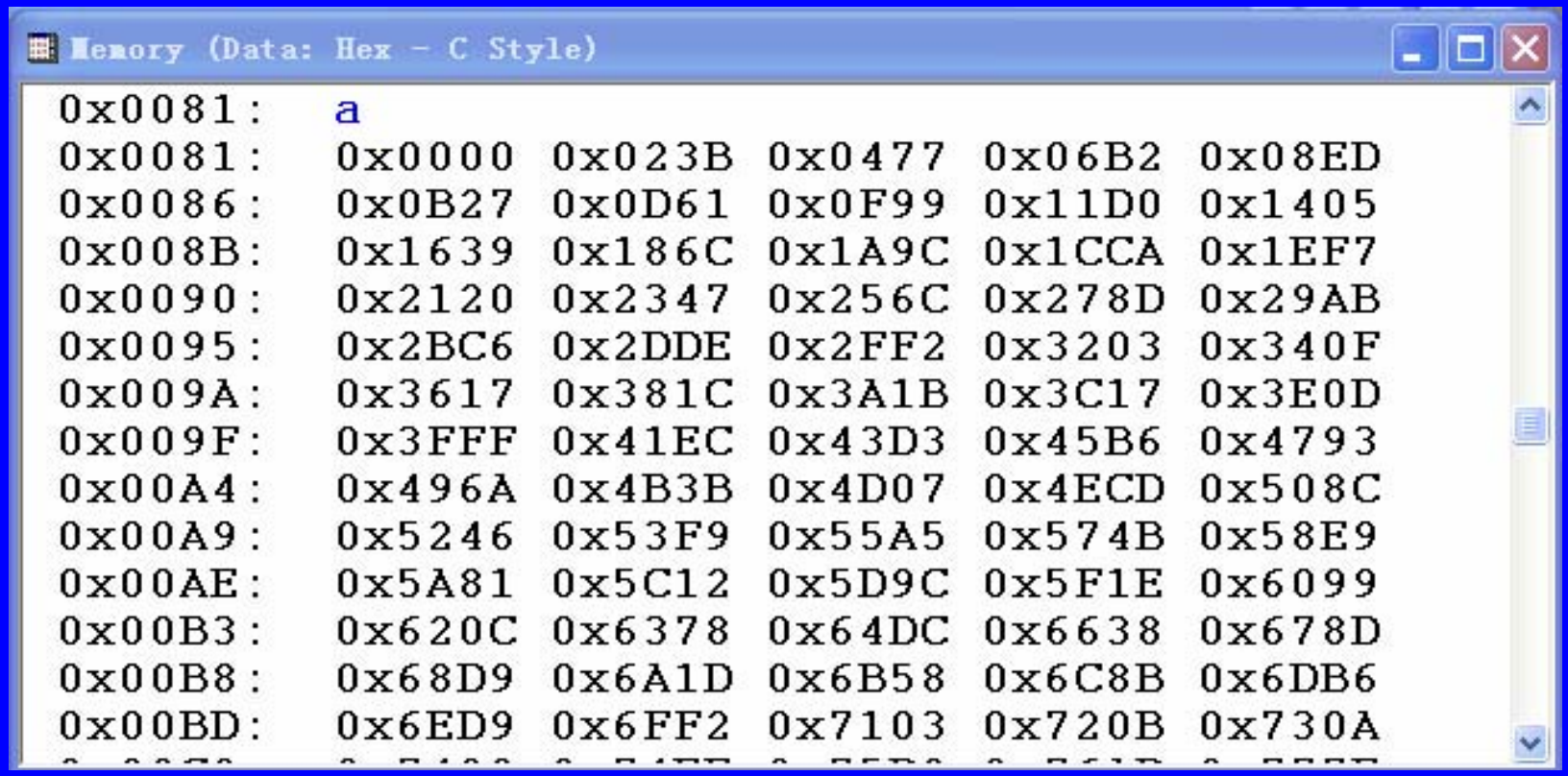


图4-19 观察到的数据存储器中的数据???

2. 静态图形显示

CCS提供了几种静态图形显示方式：

Time/Frequency	时域/频域图
Constellation	星座图
Eye Diagram	眼图
Image	图像

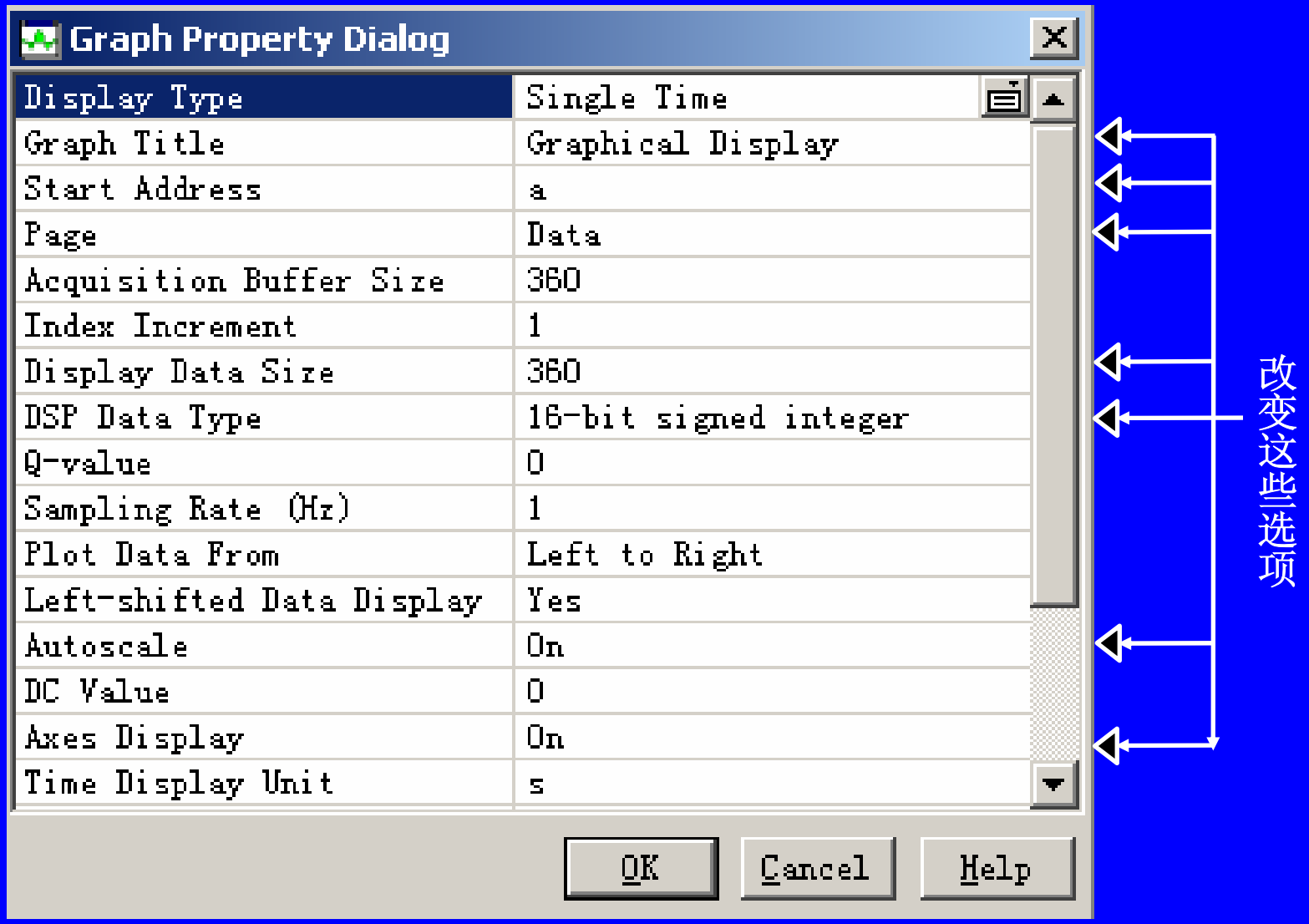


图4-20　图形显示方式属性设置对话框

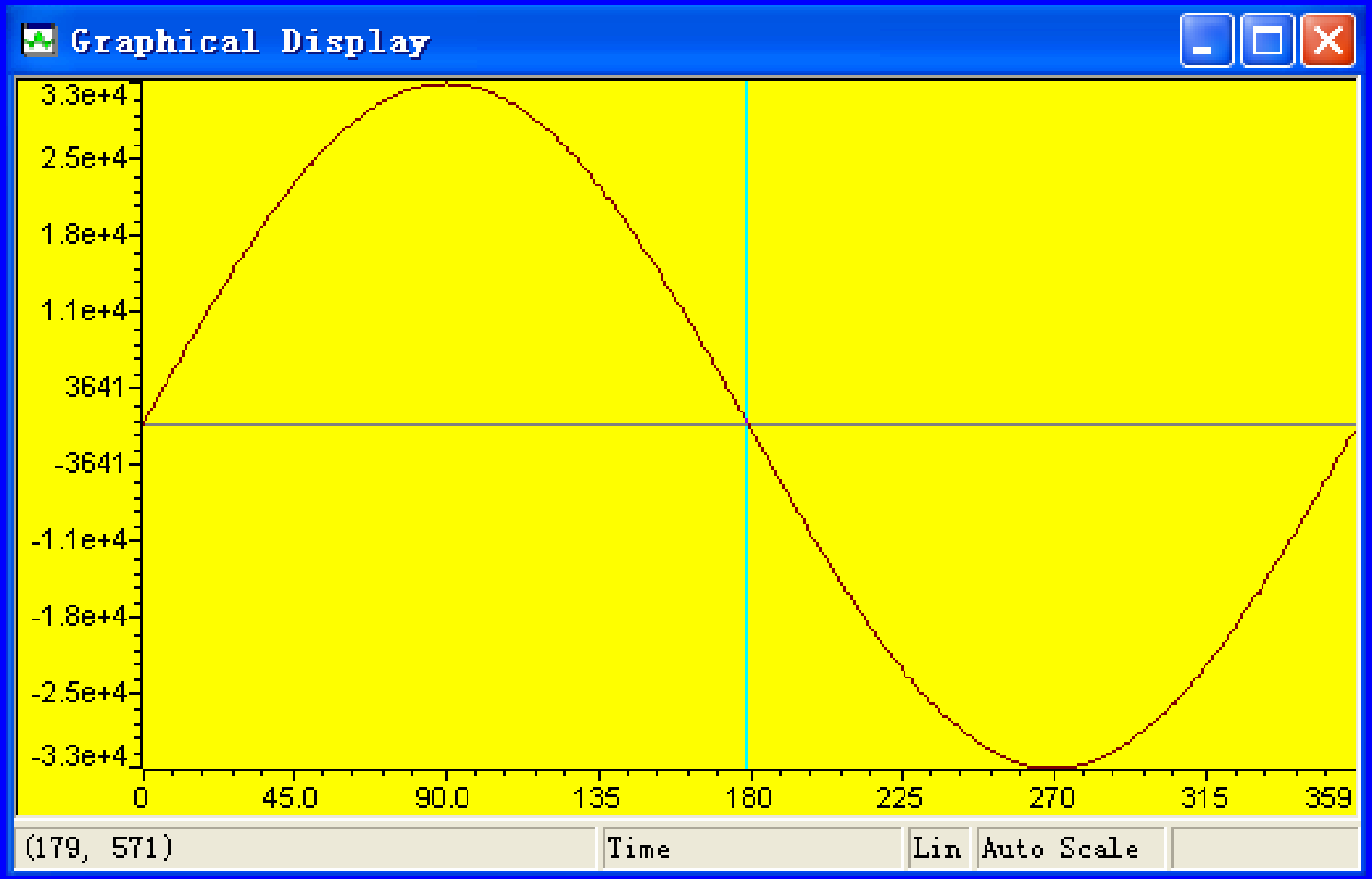

Graphical Display
3.3e+4
2.5e+4
1.8e+4
1.1e+4
3641
-3641
-1.1e+4
-1.8e+4
-2.5e+4
-3.3e+4
0
45.0
90.0
135
180
225
270
315
359
(179, 571)
Time
Lin
Auto Scale

图4-21 图形显示窗口

3. 动态图形显示

为了使数据和图形随程序运行过程发生改变，要在测试点处产生一个断点，这个断点能使图形发生变化。使用**animate**命令，程序在到达断点时，自动更新各显示窗口的内容，然后，自动恢复程序的执行。以**sine.c**为例，方法如下：

（1）在**sine.c**窗口，在语句“**a[i]=(int)(sin(i*3.14159/180)*32767);**”处，设置断点；

（2）合理安排窗口，可在屏幕上看到图形。

（3）选择**Option→Customize…**，在**Debug Properties**标签下设置动态速率（**animate speed**），**1**秒、**2**秒或者更多。控制程序运行时间，**1**秒、**2**秒或者更多时间数据刷新一次。

（4）选择**Debug→Animate**运行程序。**animate**命令与**run**命令相似，它使目标程序运行到达断点为止，这时目标程序停止，窗口被修改。

animate与**run**不同的是，**animate**命令恢复命令的执行一直到另一个断点。这个过程持续直到目标程序手动停止为止。

在点击**animate**使程序运行时，可以看到窗口的图形以一定的时间在不断的刷新。

4.4.2.5 设置探针和文件输入/输出

（1）将光标放在“**dataIO( );**”语句处，点击鼠标右键选择**Toggle Probe Point**项，此时该语句行前出现一个蓝色的菱形标志，表明探针已设置；

（2）连接输入输出文件。

① 选择**File→File I/O**，打开“**File I/O**”对话框，如图**4-22**所示。

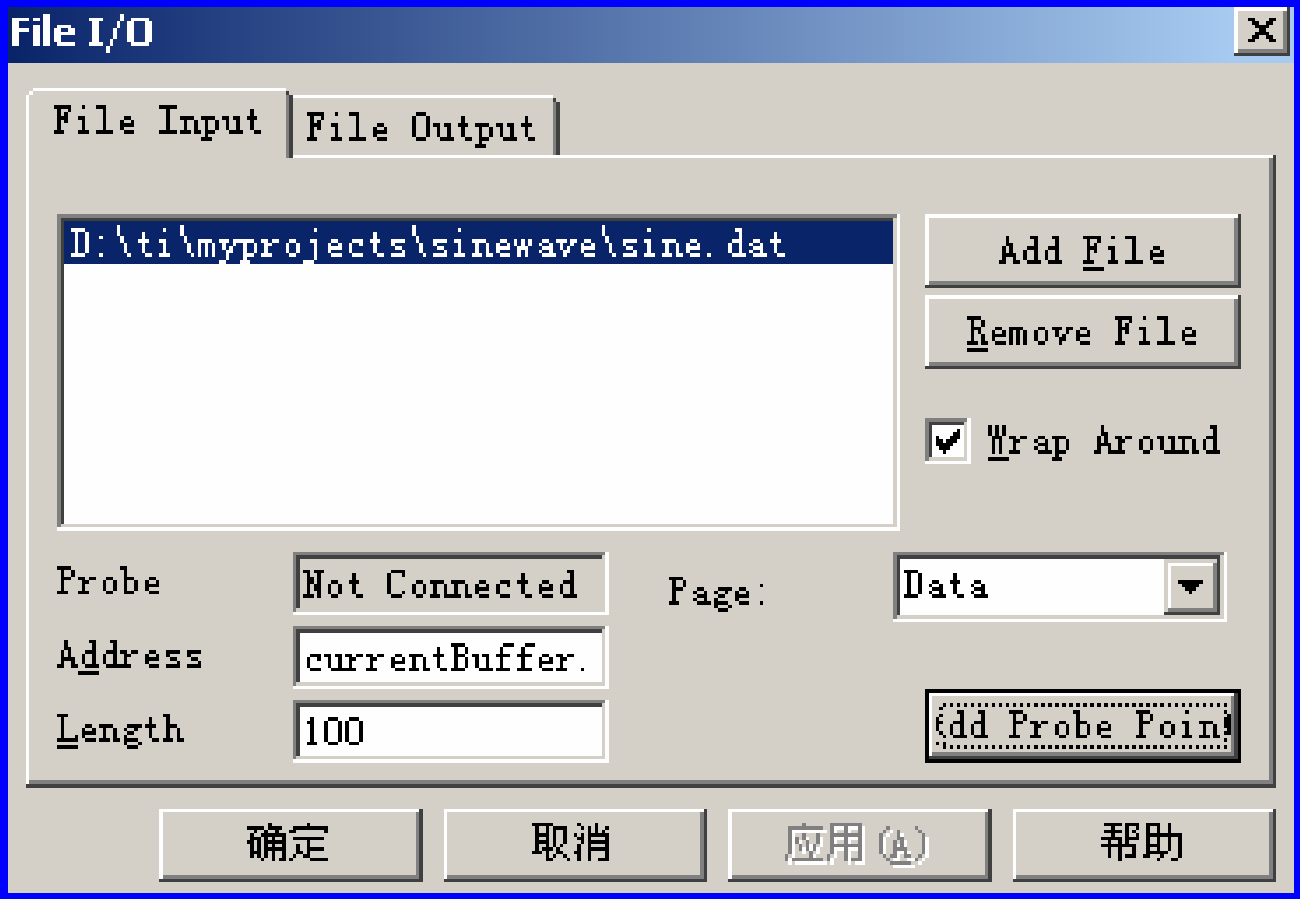


图4-22 “File I/O”对话框

② 在**File Input**表中，点击**Add File**按钮。在弹出的对话框中查找**sine.dat**文件，并将它打开，加到**“File I/O”**对话框中；若要添加或输出其他文件重复以上操作。

③ 设数据存放的存储地址及长度。地址字段指定数据传送或获取的地址，可以在该区域输入有效的符号或数字。长度区域表明每次传输或读取的抽样点数。在**“File I/O”**对话框中，将地址改为**currentBuffer.input**，长度改为**100**。

④ 在**Wrap Around box**框中打√，选择该项可以循环执行文件，当执行到结尾时，再返回顶端执行。这样对于从文件产生周期信号非常有用。如果没有选择该选项，则当到了文件结束时会有提示信息：程序终止。

⑤ 当刚把文件导入**“File I/O”**对话框时，文件还没有和探针连接。此时，**probe**区域显示**“Not Connected”**。

⑥ 点击**Add Probe Point**按钮，打开**“Break/Probe Points”**对话框，如图**4-23**所示。

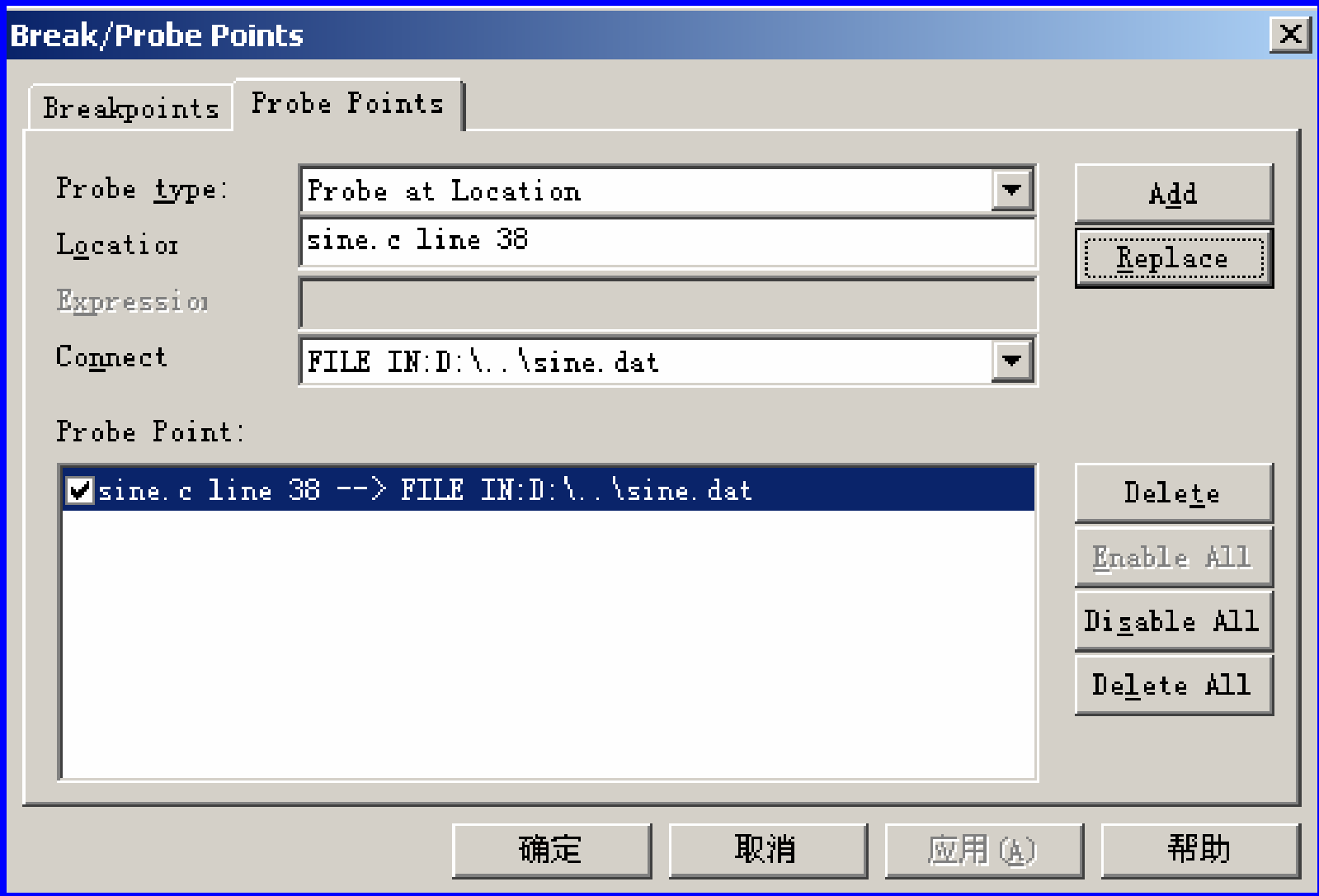


图4-23　断点/探针对话框

⑦在探针类型下拉列表中，选择所需探针。

⑧在**Probe Point**列表框中，点击**sine.c line 38-->No connect**行，此时在**Location**框中出现**sine.c line 38**，表明探针所在的地址已输入到地址框中。

⑨在**Connect**下拉列表中选择**sine.dat**文件，点击**Replace**按钮及确定按钮。

⑩在**“File I/O”**对话框中的**probe**表中可以看到**connected**，表示已经连接，点击确定按钮，文件连接成功。文件连接之后，屏幕上显示如图**4-24**所示的窗口。

图4-24 sine.dat文件控制窗口

此图下面的四个按钮分别为：开始、停止、重绕、快进。按快进按钮，**sine.dat**中的数据就会读到**currentBuffer.input**数据单元中。此时打开**memory**窗口，将起始地址改为**currentBuffer.input**，即可以看到这些数据。如图**4-25**所示

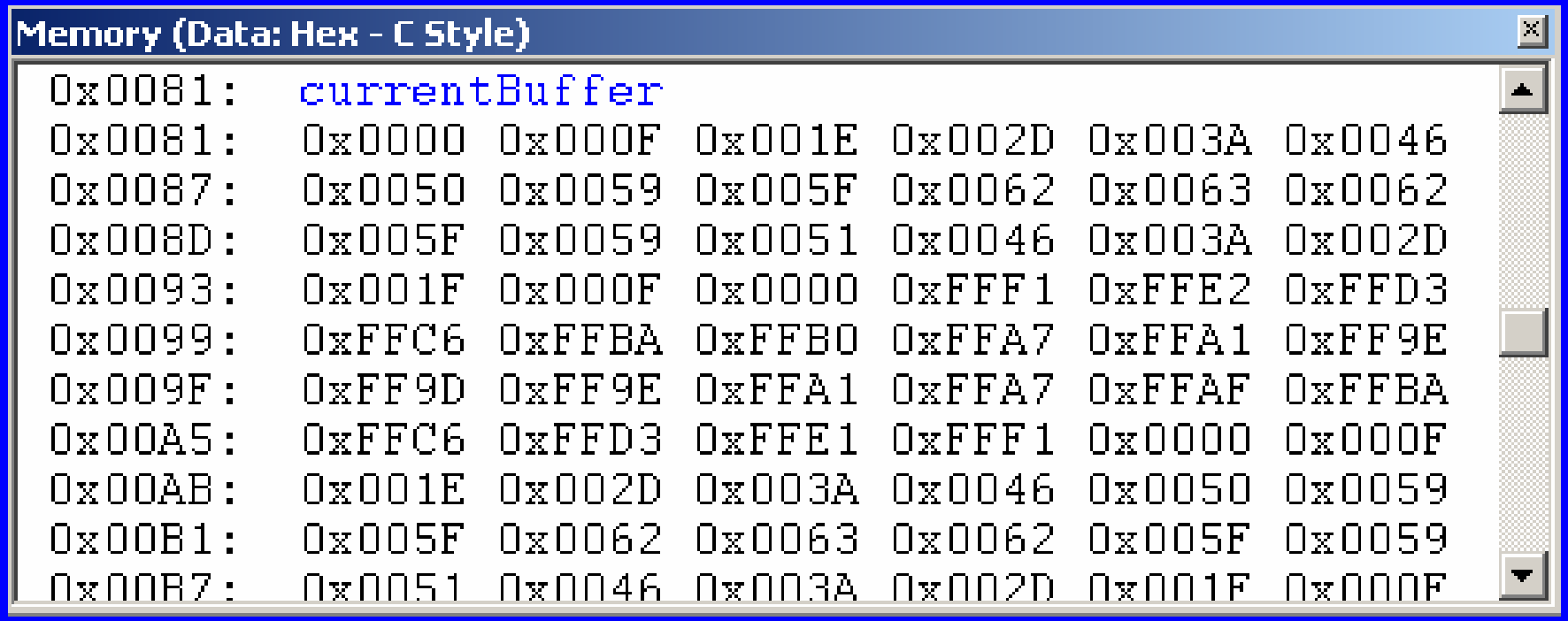


图4-25 数据导入后的数据存储器窗口

4.4.2.6 评估代码性能（Profiler的使用）

1 函数执行时间分析

2 某段程序执行时间分析

1 函数执行时间分析

（1）选择菜单命令**File→Reload Program**，重新加载程序**sinewave.out** 。

（2）选择菜单命令**Profiler→Start New Session**，在打开的对话框中输入**MySession**作为代码分析统计观察窗口的名称，然后单击**OK**按钮，则打开分析（**Profiler**）窗口，单击**Functions**标签，并点击图标**“Profile All Functions”**，出现如图**4-26**所示的画面。

（3）在工程视图中双击**volume.c**以显示文件内容。

（4）程序运行约**1**分钟后停止，看到如图**4-27**所示的分析结果。

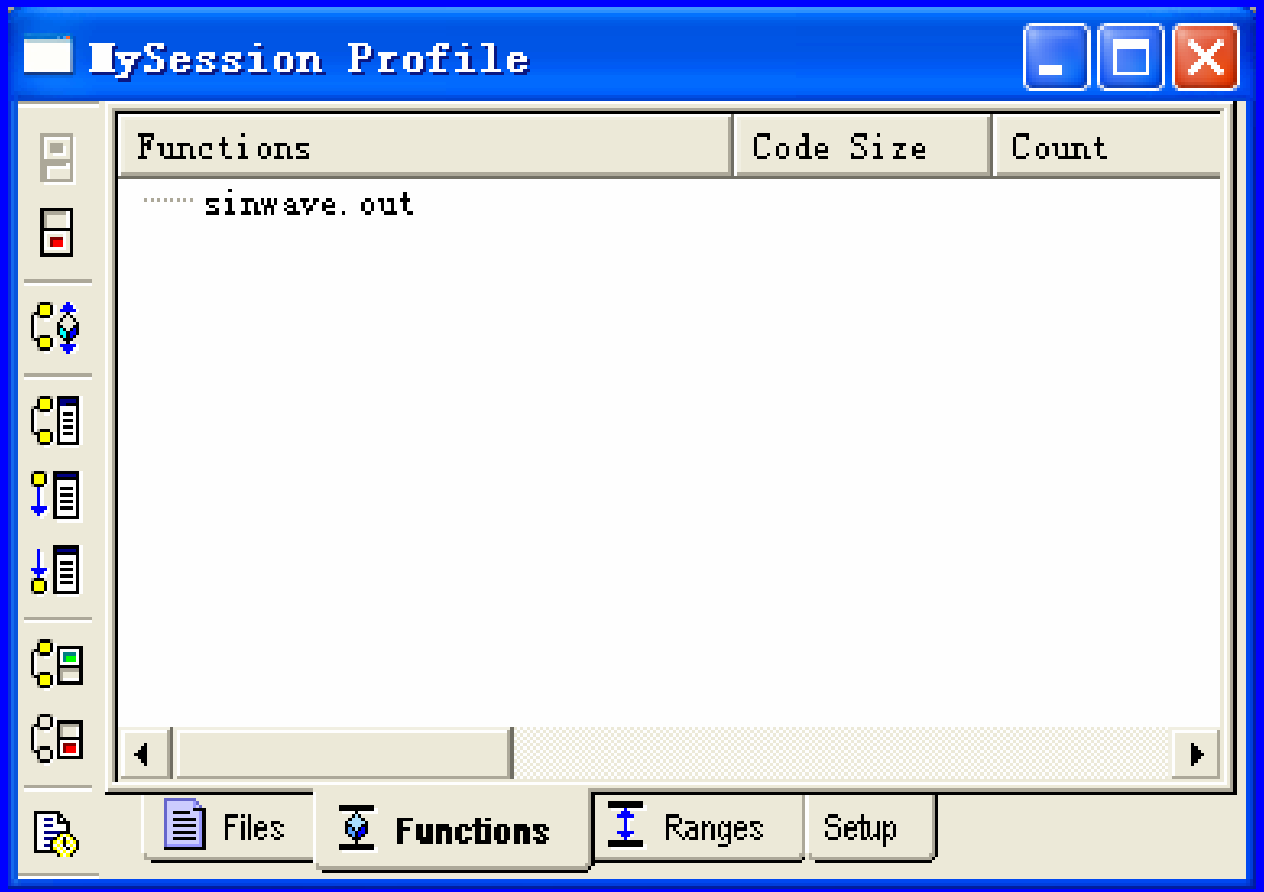


图4-26 分析窗口的Functions标签

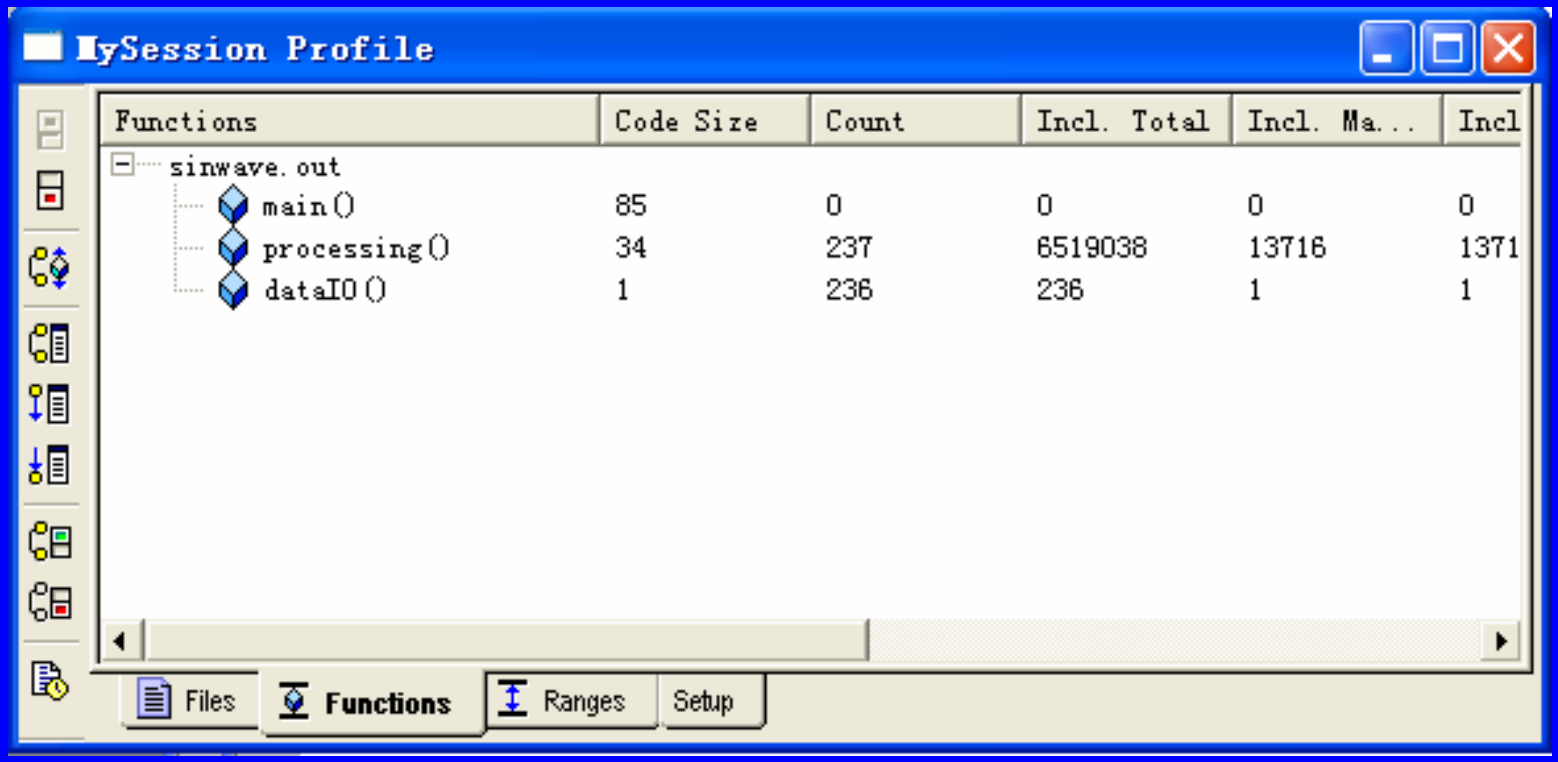


图4-27 函数执行时间分析结果

2 某段程序执行时间分析

（1）在分析窗口中单击**Ranges**标签，在工程视图中双击**sine.c**以显示源程序。

（2）将 **for(i=0;i<360;i++)**和

a[i]=gain*(int)(sin(i*3.14159/180)*32767);两行高亮显示并拖曳至分析窗口。

（3）选择菜单命令**View→Disassembly**以打开**disassembly**窗口，在**disassembly**窗口中右击，选择**Start Address**，然后输入**_c_int00**作为起始地址。

（4）在**disassembly**窗口将**_c_int00**下面的**4**行拖曳到分析窗口，如图**4-28**所示。

（5）选择菜单命令**Debug→Restart.**，然后选择**Debug→Run**。程序运行约**1**分钟后停止，看到如图**4-29**所示的分析结果。

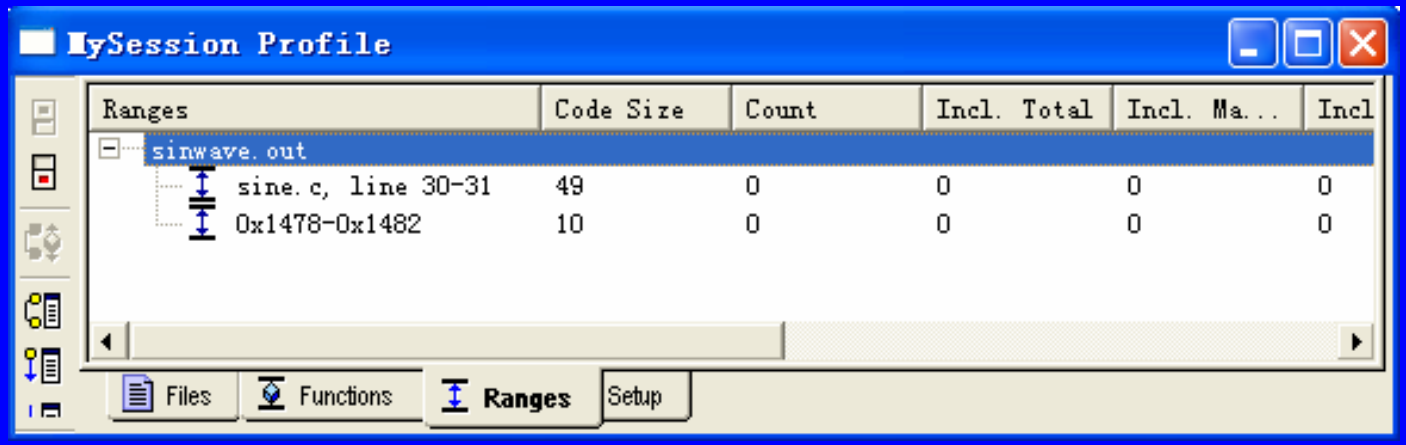


图4-28 分析窗口的Ranges标签

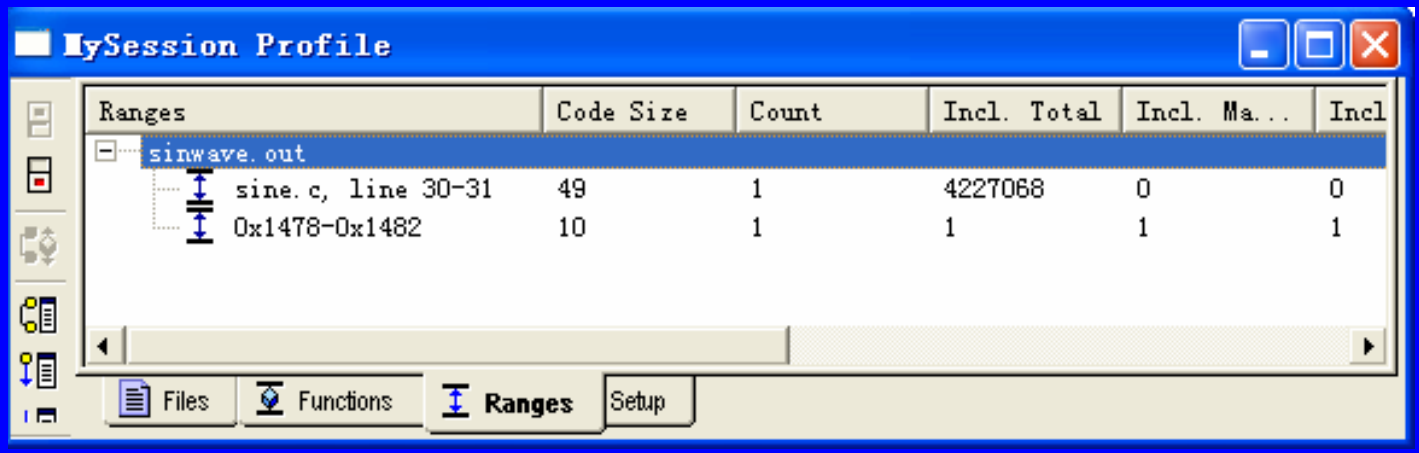


图4-29 某段程序执行时间分析结果

4.4.2.7 外部中断仿真

’C54x软件仿真器可以仿真：

外部中断信号**INT0**～ **INT3**

BIO信号

需要建立一个输入数据文件

说明何时（第几个**CPU**周期）产生这一个信号

进入软件仿真器后，要说明仿真哪个引脚信号

输入数据文件格式

[时钟周期，逻辑值]　rpt　{n|EOS}

“时钟周期”参数——表示在第几个**CPU**时钟产生，中断用绝对值或相对值表示

“逻辑值”参数 —— 仅用于仿真**BIO**引脚

rpt {n|EOS}参数——任选项，表示重复仿真中断

输入数据文件举例

[例1]　EX1.DAT

[12,1]　[23,0]　[45,1]

第12个时钟周期，　BIO=高电平
第23个时钟周期，　BIO=底电平
第45个时钟周期，　BIO=高电平

[例2]　EX2.DAT

5　(+10　+20)　rpt　2

表示要在第**5，15，35，45，65**个时钟周期仿真中断。

括号内为重复部分。

[例3]　EX3.DAT

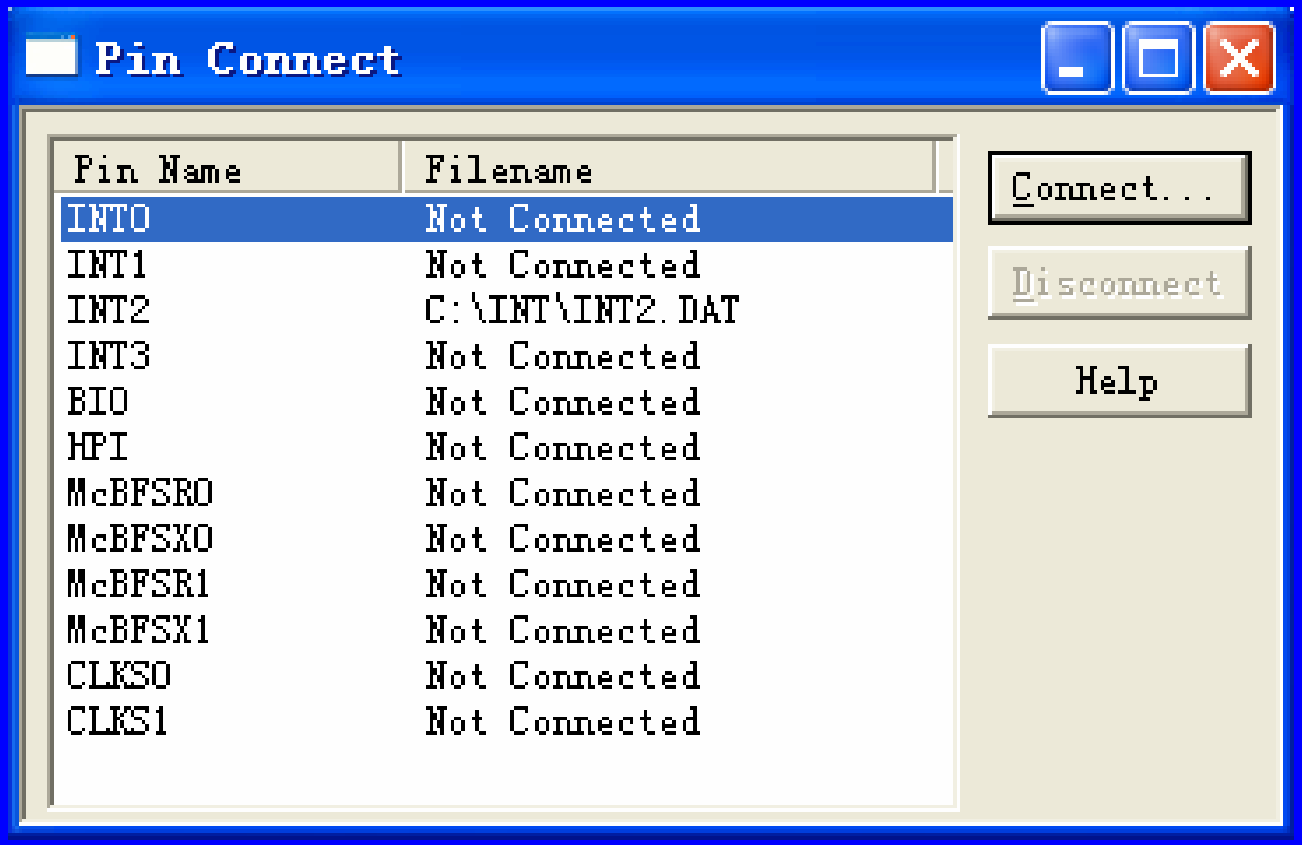
10　(+5　+20)　rpt　EOS

表示要在第**10，15，35，40，60……**个时钟周期仿真中断，直到结束。

括号内为重复部分。

仿真DSP引脚与输入数据文件相连

进入CCS后，选择Tools →Pin Connect，加进INT2仿真数据文件后，得到如下窗口：



4.4.3 GEL工具

GEL是通用扩展语言（General Extension Language）的简称，是一种类似于C语言的解释性语言。用GEL可以建立需要的函数来扩展CCS的功能。GEL语言有它相对应的语法，遵循这些语法，可以建立GEL函数。然后将GEL函数装入CCS中，去访问实际的或仿真的目标存储器。

GEL函数定义如下：

FuncName([参数1]，[参数2]……)

{

语句

}

其中：

FuncName：定义的**GEL**函数名

参数：有效的**GEL**参数

语句：有效的**GEL**语句

表4-11 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1。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

表4-12 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( )**

为了使用**GEL**工具，必须创建包含**GEL**函数的文件（**.gel**文件）。当创建了**GEL**文件之后，**GEL**文件必须载入到**CCS**之中才能够访问这个文件中的函数。这样，**GEL**函数就位于**CCS**的内存中，而且可以在任何时刻被执行；除非**GEL**文件从**CCS**之中移走。如果已经载入的**GEL**函数进行了修改，必须把它卸载之后再次载入，所做的修改才会生效。

载入**GEL**文件非常简单，其步骤如下：选择**File→Load GEL**；也可以在**project**窗口的**GEL**文件夹的位置单击右键，算则**Load GEL**。在“载入**GEL**文件”对话框当中，指定包含所需**GEL**函数的文件（***.gel**）。

4.4.3.1 利用GEL工具定义增益滑块

```
        menuitem "Gain_Control";
slider gain(1,10,1,1,volume)
{
  /*contrl the target variable "gain" with the
    parameter
  "Gain_Control" passed by the slider object.*/
  gain=volume;
}
```

4.4.3.2 利用GEL工具实现数据文件与I/O端口的链接

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。

GEL文件中的端口连接函数如下：

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。