Code Composer Studio
Getting Started Guide

PRINTED WITH
SOY INK™

TEXAS
INSTRUMENTS

Printed on Recycled Paper

Read This First

About This Manual

To get started with Code Composer Studio ™ (CCS) IDE, you must go through the first two chapters of this book. The remaining chapters contain information that can be useful to you, depending on your needs and the tools you are using. To determine whether you can utilize the features in these chapters, please review the online help provided with Code Composer Studio.

How to Use This Manual

This document contains the following chapters:

Chapter	Title	Description
1	Getting Started with Code Composer Studio IDE	Walks you through the steps of setting up the CCS IDE and shows you how to access documentation.
2	Code Composer Studio Project Management and Editing Tools	Instructions on the basic functionality of the CCS IDE.
3	Code Composer Studio Code Generation Tools	Reviews development tools provided with CCS IDE.
4	Code Composer Studio Debug Tools	Reviews the available debug tools and the device generations to which they apply.
5	Code Composer Studio Optimization Tools	Provides information on profiling code and the 'C6000 PBC.
6	Code Composer Studio Real Time Components	Overview of the real-time components featured in CCS IDE.
7	Code Composer Studio Chip Support Library Overview	Provides and overview of the features and architecture of CSL.
8	TMS320 DSP Algorithm Standard	Reviews the TMS320 DSP Algorithm Standard, and explains the resources available to algorithm writers.

Notational Conventions

This document uses the following conventions.

❏ Program listings, program examples, and interactive displays are shown in a `special typeface` similar to a typewriter's. Examples use a **`bold version`** of the special typeface for emphasis; interactive displays use a **`bold version`** of the special typeface to distinguish commands that you enter from items that the system displays (such as prompts, command output, error messages, etc.).

Here is a sample program listing:

```
0011   0005   0001          .field    1, 2
0012   0005   0003          .field    3, 4
0013   0005   0006          .field    6, 3
0014   0006                 .even
```

Here is an example of a system prompt and a command that you might enter:

```
C:   csr –a /user/ti/simuboard/utilities
```

Related Documentation From Texas Instruments

For additional information on your target processor and related support tools, see the online manuals provided with the CCS IDE.

To access the online manuals:

Help→CCS Documentation→Manuals

Related Documentation

You can use the following books to supplement this user's guide:

American National Standard for Information Systems-Programming Language C X3.159-1989, American National Standards Institute (ANSI standard for C)

The C Programming Language (second edition), by Brian W. Kernighan and Dennis M. Ritchie, published by Prentice-Hall, Englewood Cliffs, New Jersey, 1988

Programming in C, Kochan, Steve G., Hayden Book Company

Trademarks

Code Composer Studio, DSP/BIOS, Probe Point(s), RTDX, TMS320C6000, and TMS320C5000 are trademarks of Texas Instruments Incorporated.

Pentium is a registered trademark of Intel Corporation.

Windows and Windows NT are registered trademarks of Microsoft Corporation.

All trademarks are the property of their respective owners.

To Help Us Improve Our Documentation . . .

If you would like to make suggestions or report errors in documentation, please email us. Be sure to include the following information that is on the title page: the full title of the book, the publication date, and the literature number.

Email: support@ti.com

Contents

Figures

Getting Started with
Code Composer Studio IDE

This chapter applies to all platforms using Code Composer Studio™ (CCS) IDE.

This chapter gives you a short overview of the CCS development flow, and then walks you through the steps of setting it up. It also shows you how to use the CCS IDE and access the documentation.

1.1 Development Flow

Understanding the development flow helps you understand how to use the different components of the CCS IDE.

The CCS IDE extends the basic code generation tools with a set of debugging and real-time analysis capabilities. The CCS IDE supports all phases of the development cycle shown here:

Figure 1–1. Simplified CCS Development Flow

Design	Code & build	Debug	Analyze
conceptual planning	create project, write source code, configuration file	syntax checking, probe points, logging, etc.	real-time debugging, statistics, tracing
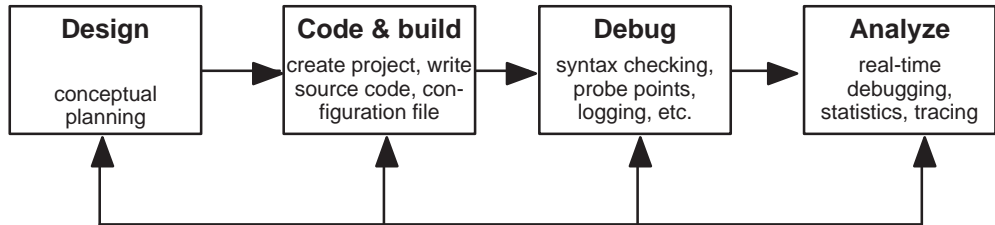
1.2 Creating a System Configuration

CCS Setup allows you to configure the CCS IDE software to work with different hardware or simulator targets. You can quickly begin working using the default configuration or one of the standard configuration files supplied with CCS IDE. For the C5000™ system, the default configuration is the C55x™ simulator, and for the C6000™ system, the C64x™ simulator is the default configuration.

CCS Setup provides you with the option of creating a configuration using standard configuration files, or creating a customized configuration using your own configuration files. For the purposes of this example, the standard configuration files are used. (If you want to create a customized system configuration file, see the online help and/or the tutorial provided with the CCS product.)

To create a system configuration using a standard configuration file:

Step 1: Start CCS Setup by double clicking on the Setup CCS desktop icon.

Step 2: Click the Clear button in the Import Configuration dialog box to remove any previously defined configuration.

Clear button removes previous configuration
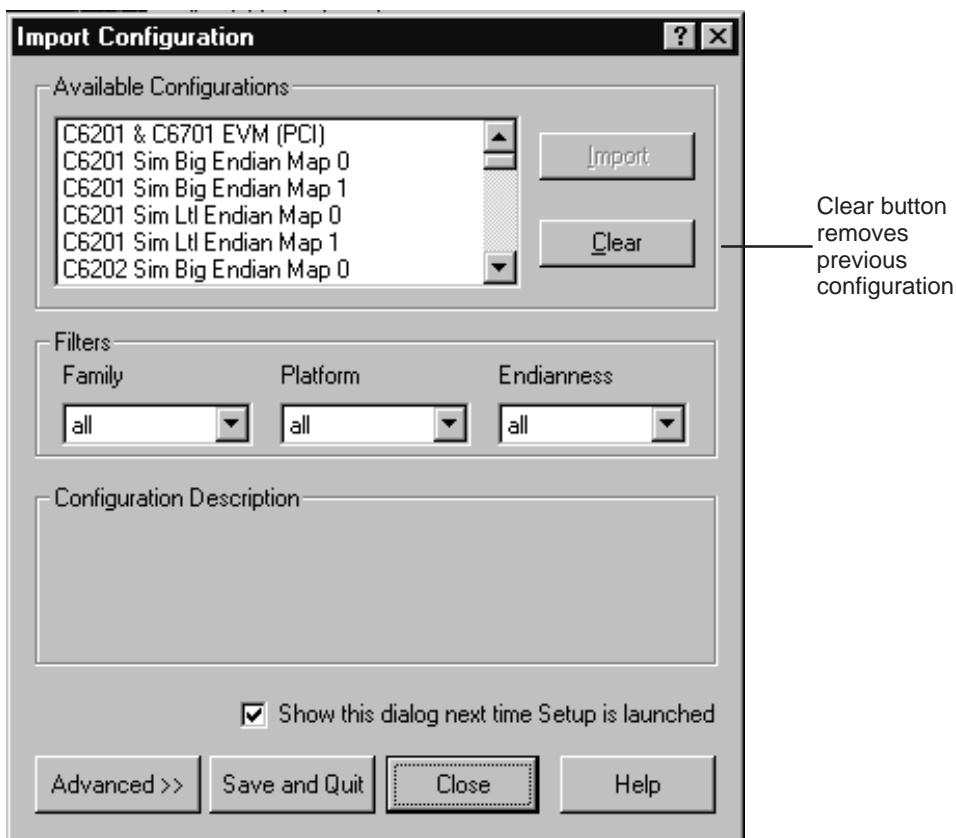
Step 3: Click Yes to confirm the Clear command.

Step 4: Select the standard configuration that matches your system from the list of Available Configurations.

Standard Configurations

Description of highlighted configuration

> **Import Configuration** [?] [X]
>
> ┌─ Available Configurations ──────────────────────┐
> │ C6201 & C6701 EVM (PCI) ▲ │ [Import]
> │ C6201 Sim Big Endian Map 0 │
> │ C6201 Sim Big Endian Map 1 │
> │ C6201 Sim Ltl Endian Map 0 │
> │ C6201 Sim Ltl Endian Map 1 │ [Clear]
> │ C6202 Sim Big Endian Map 0 ▼ │
> └───┘
>
> ┌─ Filters ───────────────────────────────────────┐
> │ Family   Platform   Endianness │
> │ [all   ▼]   [all   ▼]   [all   ▼] │
> └───┘
>
> ┌─ Configuration Description ─────────────────────┐
> │ One TMS320C6xxx CPU on an EVM board. │
> │ │
> └───┘
>
> ☑ Show this dialog next time Setup is launched
>
> [Advanced >>] [Save and Quit] [Close] [Help]

Read the information displayed in the Configuration Description portion of the dialog box to help you determine if one of the available configurations matches your system.

If none of the standard configurations adequately describe your system, you must create a customized configuration (see the online help and/or the tutorial provided with the CCS product).

Step 5: Click the Import button to import your selection to the system configuration currently being created in the CCS Setup window.

The configuration you selected now displays under the My System icon in the System Configuration pane of the Setup window.

If your configuration has more than one target, repeat steps 4 and 5 until you have selected a configuration for each board.

Step 6: Click the Save and Quit button to save the configuration in the System Registry.

Step 7: Click the Yes button to start the CCS IDE when you exit CCS Setup.

The CCS Setup closes and the CCS IDE automatically opens using the configuration you just created.
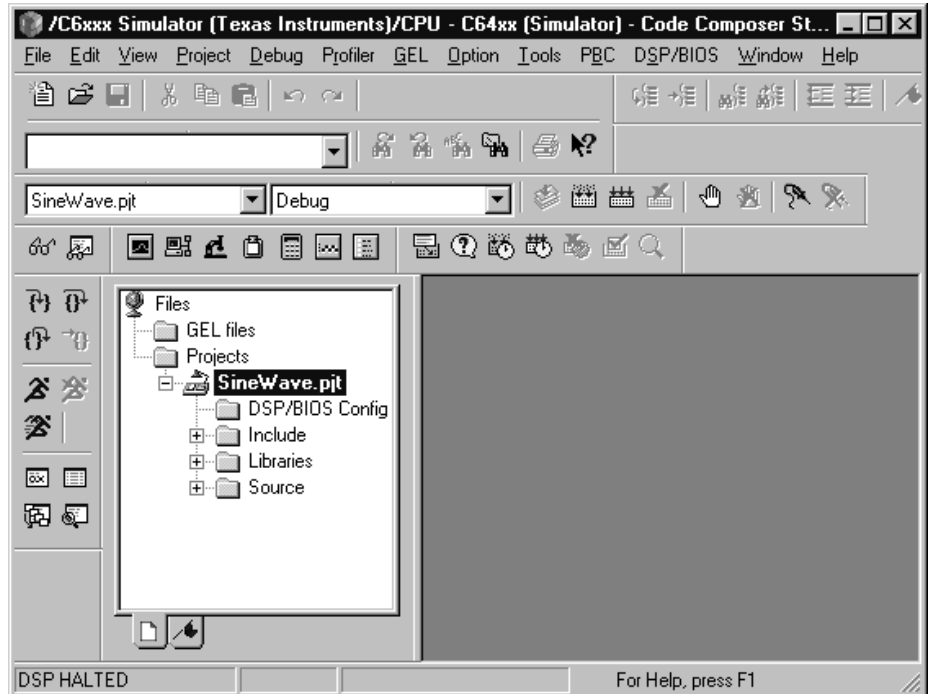
You can now start a project in the CCS IDE.

See Chapter 2 of this book, or the online help and tutorial provided with the CCS IDE, for information on starting a project.

1.3 Getting Started with CCS Tutorial

When you have completed the installation and setup process, run the CCS Tutorial. This tutorial familiarizes you with the CCS features, including what is new in this version. Performing this tutorial before you use the CCS IDE can help shorten your learning time and provides information on many fundamental procedures.

To access the CCS Tutorial, follow these steps:

1) Start the CCS IDE by double-clicking on the "CCS 2" icon located on the desktop.

2) From the CCS Help menu, select Tutorial.

1.4   Accessing CCS Documentation

The CCS online help provides access to platform-specific documentation. To open the CCS Help, select Help→Contents. The Welcome page of the CCS Help displays. The table below lists the links on the Welcome page that provide important information:

For this information...	click this link.
New features available in CCS IDE v2	What's New
CCS IDE v2 release notes	Release Notes
User's guides, reference guides, and application reports (in PDF format)	Online Manuals

Use F1 to obtain help on components within the CCS interface:

❏ To obtain the description of an instruction or register while editing your source code in a CCS document window, double-click or click-and-drag to highlight an instruction name or register name, then press F1.

❏ To obtain the description of a CCS window or dialog box, click in the window or dialog box to make it active, then press F1.

❏ To obtain the description of a menu bar or toolbar command, pause the cursor over the command, then press F1.

Accessing Documentation from the Start Menu

To access the customer support guides, license agreement, and the CCS online help:

Step 1:   From the Start menu, choose Start → Programs → Texas Instruments → Code Composer Studio 2 → Documentation.

Step 2:   Select the document you want to view.

1.5   Update Advisor

The Update Advisor allows you to download updated versions of the CCS IDE and related tools. The Update Advisor accesses the Available Updates web site. This site displays a list of CCS patches, drivers, and tools available for downloading.

Note:

To use the Update Advisor, you must have Internet access and a browser installed on your machine. See the CCS IDE Quick Start for complete system requirements.

You must register online and have a valid subscription plan in place to receive downloads through update advisor. You receive a 90 day free subscription service with the CCS product. At the end of this period, you must purchase an annual subscription service. Annual subscriptions are only available for the full CCS product.

If you did not register your product during installation, you can access the online registration form from the CCS help menu: Help→CCS on the Web→ Register.

To Check for Tool Updates

In the CCS IDE, select Help→CCS on the Web→Update Advisor.

Important! The first time you use Update Advisor, your browser may display the TI&ME web page. You must be registered with TI&ME before you can access the Available Updates web site. To register, follow the directions displayed on the page. If you are already registered with TI&ME, and have accepted the cookie necessary for automatic log-in, your browser will go directly to the Available Updates web site.

In order to query the Available Updates web site, the Update Advisor passes certain information from your machine:

❑   CCS product registration number
❑   CCS installation version
❑   a text description of the installed product
❑   the list of installed plug-ins

The Available Updates web site will then list any updates appropriate for your CCS installation.

You have the opportunity to just download the updates, or to download and install them immediately.

You can also configure the Update Advisor to automatically check for updates.

To Uninstall the Updates

Any installed update can be uninstalled to restore the previous version of the CCS IDE.

Note that only the previous version of a tool can be restored. If you install one update for a tool, and then install a second update for the same tool, the first update can be restored. The original version of the tool cannot be restored, even if you uninstall both the second update and the first update.

To Automatically Check for Tool Updates

With the Update Advisor, you may check for tool updates at any time, or you can configure the Update Advisor to automatically check for updates.

Step 1: Select Help→CCS on the Web→Update Setting. The Web Settings dialog box appears:
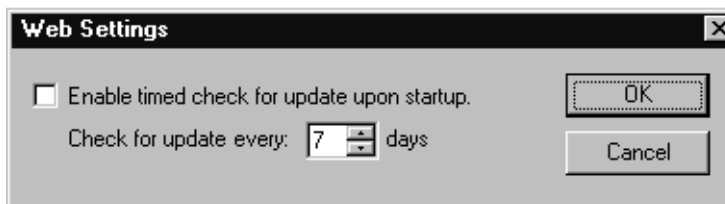
Step 2: In the Check for Update field, specify how often the Update Advisor should check the Available Updates web site.

Step 3: To enable the automatic update feature, click the checkbox to the left of the "Enable timed check for update upon startup" field.

When this field is enabled, the Update Advisor automatically checks for web updates according to the schedule specified in step 2.

Step 4: Click OK to save your changes and close the dialog box.

1.6   Component Manager

Multiple installations of the CCS IDE can share installed tools. The Component Manager provides an interface for handling multiple versions of tools with multiple installations of the CCS IDE.

The Component Manager window displays a listing of all CCS installations, build tools, Texas Instruments plug-in tools, and third-party plug-in tools. When a node is selected in the tree (the left pane of the Component Manager), its properties are displayed in the Properties pane (the right pane).

With the Component Manager, you can enable or disable tools for a particular CCS installation. This functionality allows you to create a custom combination of tools contained within a CCS system. The Component Manager also allows you to access the Update Advisor to download the most recent version of the tools from the web. To use the Update Advisor, you must have Internet access and a browser installed on your machine.

Note:

The component manager is an advanced tool use primarily to customize or modify your installation. Use this tool only to resolve component interaction in a custom or multiple installation environment.

Figure 1–2. Component Manager

Tree listing of all CCS
installations and tools

Properties of the item highlighted
in the CCS installation pane
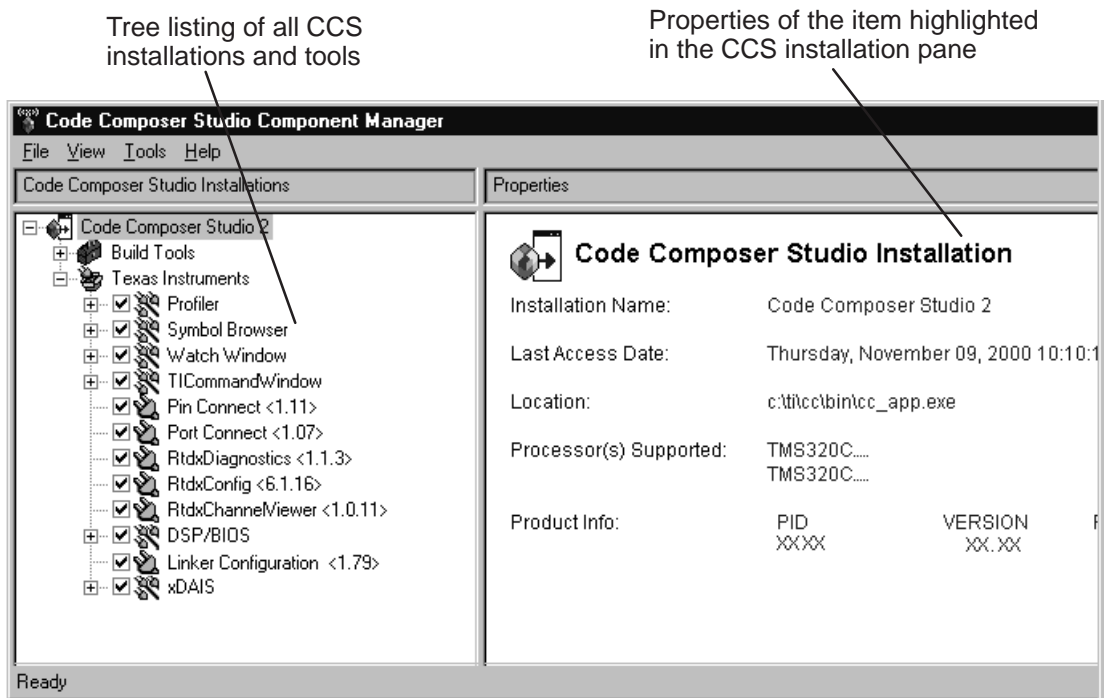
Opening Component Manager

To open the Component Manager:

Step 1: From the Help menu in the CCS IDE, select About.

The About CCS dialog box appears.

Step 2: In the About dialog box, click the Component Manager button.

The Component Manager window displays.

Multiple Versions of the CCS IDE

The following is a list of requirements for maintaining multiple versions of the CCS IDE and related tools:

❑ If you install an additional version of the CCS IDE, or an additional version of a tool, in the same directory as its previous installation, the original installation will be overwritten. To keep more than one version of the CCS IDE or a related tool, you must install each version in a different directory.

❑ You cannot enable multiple versions of the same tool within one CCS IDE installation.

Code Composer Studio
Project Management and Editing Tools

This chapter applies to all platforms using Code Composer Studio™
(CCS) IDE.

This chapter reviews the tools and options available to you for creating and managing your programs and projects. For more information, see the online help and online manuals provided with the CCS IDE.

2.1 Creating a New Project

The information for a project is stored in a single project file (*.pjt). Use the following procedure to create new projects, one at a time. When multiple projects are created, each project's filename must be unique .

TIP: It is possible to have multiple projects open at the same time.

Step 1: From the Project menu, choose New.

The Project Creation wizard window displays.
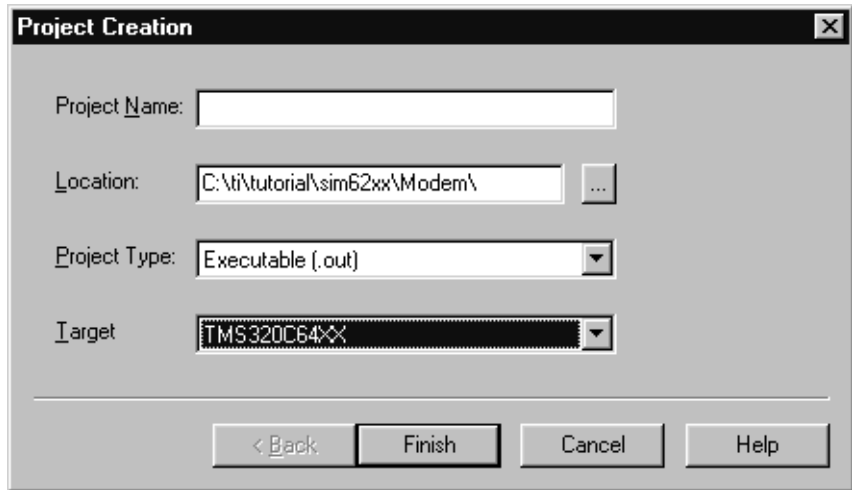
Step 2: In the Project Name field, type the name you want for your project.

Each project you create must have a unique name.

Step 3: In the Location field, specify a directory to store the project file.

You can type the full path in the Location field or click the Browse button and use the Choose Directory dialog box.

It is a good idea to use a different directory for each new project. Use this directory to store project files and the object files generated by the compiler and assembler.

Step 4: In the Project Type field, select a Project Type from the drop-down list.

Choose either Executable (.out) or Library (lib). Executable indicates that the project generates an executable file. Library indicates that you are building an object library.

Step 5: In the Target field, select the Target Family that identifies your CPU. This information is necessary when tools are installed for multiple targets.

Step 6: Click Finish.

The CCS IDE creates a project file called *projectname*.pjt. This file stores your project settings and references the various files used by your project.

The new project automatically becomes the active project. The first project configuration (in alphabetical order) is set active. The new project inherits TI supplied default compiler and linker options for debug and release configurations. For information on how to change these settings, see section 2.5, Selection a Project Configuration, found on page 2-10.

After creating a new project file, add the filenames of your source code, object libraries, and linker command file to the project list.

Figure 2–1. CCS IDE Basic Window
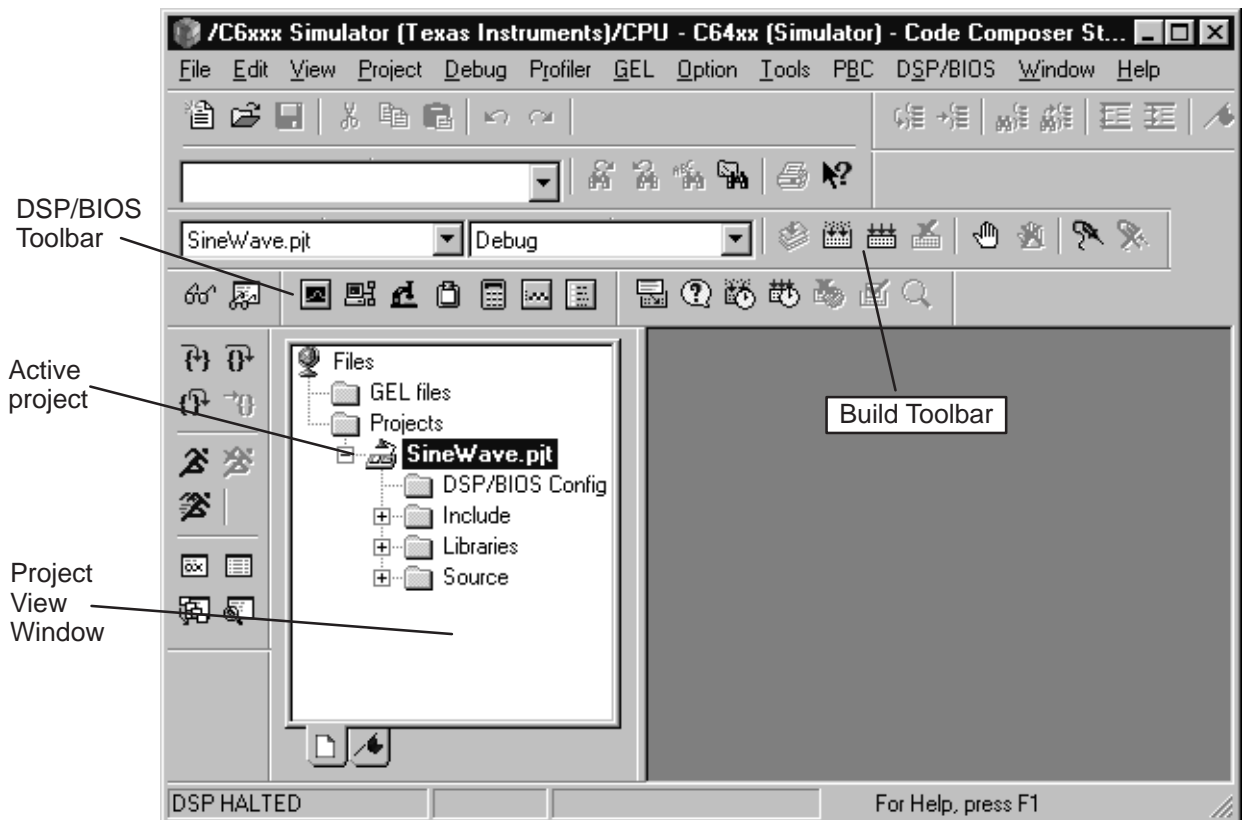
2.2 Adding Files to a Project

Step 1: Select Project→Add Files to Project, or or right-click on the project's filename in the Project View window and select Add Files.

The Add Files to Project dialog box displays.

```
Add Files to Project                                    ? X

Look in:  volume1                           ▼  🗁  ⬜  🗂  ☷ ☰

📁 Debug
🇨 volume.c




File name:  [                              ]        Open

Files of type: [C Source Files (*.c;*.ccc)      ▼]    Cancel
               C Source Files (*.c;*.ccc)
               C++ Source Files (*.cpp;*.cc;*.cxx)    Help
               Asm Source Files (*.a*)
               Linear Source Files (*.s*)
               Object and Library Files (*.o*;*.l*)
               Configuration File (*.cdb)
               Linker Command File (*.cmd)
               All Files (*.*)
```

Drop-down list
of file types
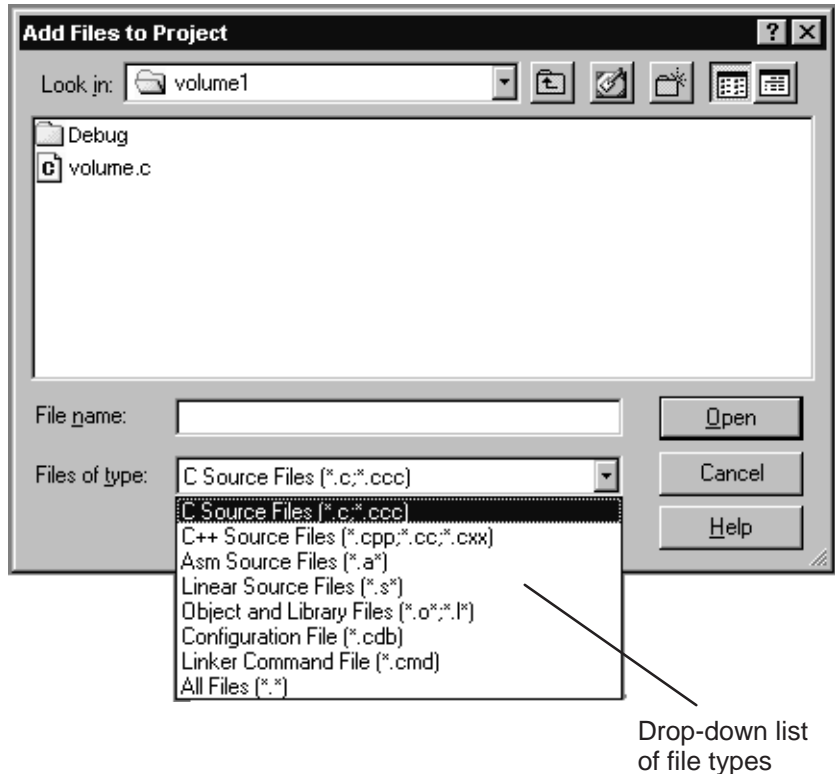
Step 2: In the Add Files to Project dialog box, specify a file to add. If the file does not exist in the current directory, browse to the correct location. Use the Files of type drop-down list to set the type of files that appear in the File name field.

Note:

Do not try to manually add header/include files (*.h) to the project. These files are automatically added when the source files are scanned for dependencies as part of the build process.

Step 3: Click Open to add the specified file to your project.

The Project View is automatically updated when a file is added to the current project.

Figure 2–2. Project View

Active project

Project View Window
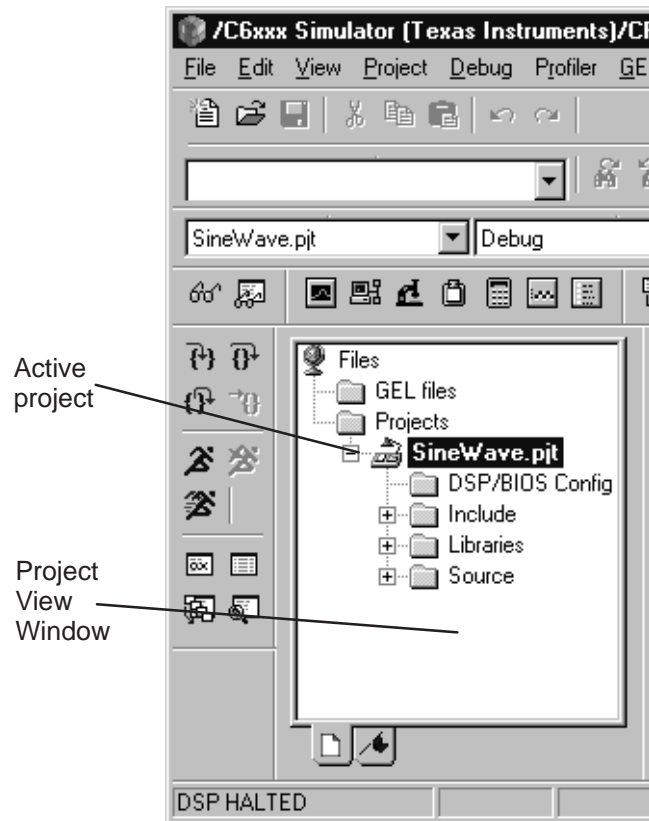
Note:

The project manager organizes files into folders for source files, include files, libraries, and DSP/BIOS configuration files. Source files that are generated by DSP/BIOS are placed in the Generated files folder.

If you need to remove a file from the project, right-click on the file in the Project View and choose Remove from project in the pop-up menu.

When building the program, the CCS IDE finds files by searching for project files in the following path order:

❏ The folder that contains the source file.

❏ The folders listed in the Include Search Path for the compiler or assembler options (from left to right).

❏ The folders listed in the definitions of the optional DSP_C_DIR (compiler) and DSP_A_DIR (assembler) environment variables (from left to right).

2.3 Using Source Control

The project manager enables you to connect your projects to a variety of source control providers. The CCS IDE automatically detects compatible providers that are installed on your computer.

Step 1: From the Project menu, choose Source Control.

Step 2: From the Source Control submenu, choose Select Provider...
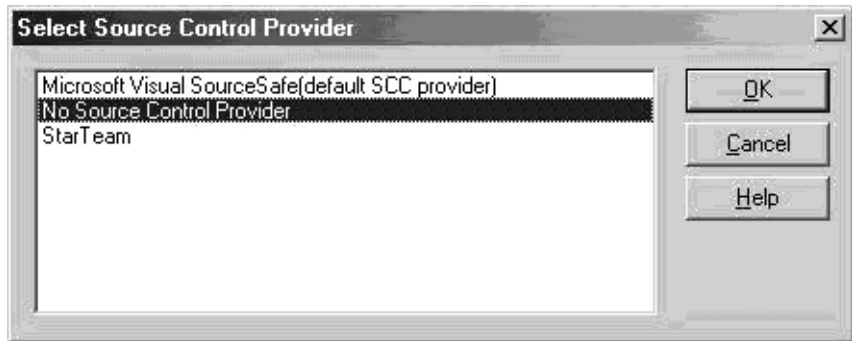
Step 3: Select the Source Control Provider that you want to use and press OK.

NOTE: If no source control providers are listed, please ensure that you have correctly installed the client software for the provider on your machine.

Step 4: Open a project and select Add to Source Control from Project→Source Control.

Step 5: Add your source files to Source Control.

You can check files in and out of source control by selecting a file in the Project View window and right clicking on the file.

Figure 2–3. Source Control Pop-Up Menu
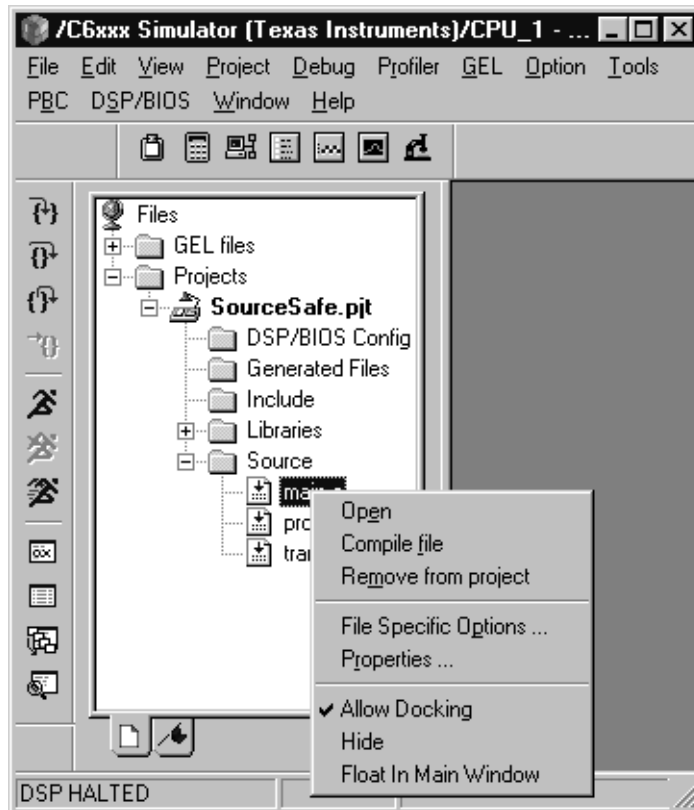
2.4 Building and Running the Program

To build and run a program, follow these steps:

TIP: You can use the supplied "timake.exe" utility to build a CCS project from the DOS shell. See section 2.6, Building Projects From the Command Line, found on page 2-11, for more information on timake.exe.

Step 1: Choose Project→Rebuild All or click the 🗓 (Rebuild All) toolbar button.

The CCS IDE recompiles, reassembles, and relinks all the files in your project. Messages about this process are shown in a frame at the bottom of the window.

Step 2: By default, the .out file is built into a debug directory located under your current project folder. To change this location, select a different one from the CCS toolbar.

Name of the current project

Project Configuration

| volume1.pjt | ▼ | Debug | ▼ | 🕮 🕮 🕮 🕮 | 🖐 🕮 | 🔍 🔍 |
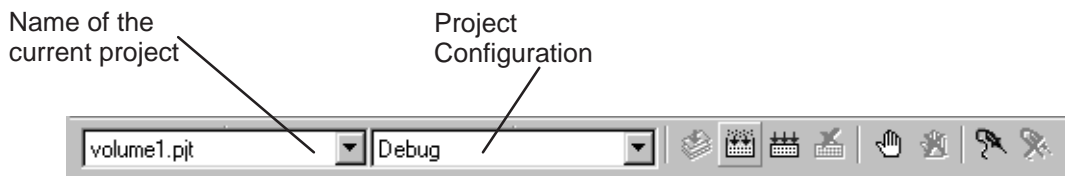
Step 3: Choose File→Load Program.

Select the program you just rebuilt, and click Open.

The CCS IDE loads the program onto the target DSP and opens a Dis-Assembly window that shows the disassembled instructions that make up the program. (Notice that the CCS IDE also automatically opens a tabbed area at the bottom of the window to show the output that the program sends to stdout.)

Step 4: Choose View→Mixed Source/ASM.

This allows you to simultaneously view your c source and the resulting assembly code .

Step 5: Click on an assembly instruction in the mixed-mode window. (Click on the actual instruction, not the address of the instruction or the fields passed to the instruction.)

Press the F1 key. The CCS IDE searches for help on that instruction. This is a good way to get help on an unfamiliar assembly instruction.

Step 6: Choose Debug→Go Main to begin execution from the main function. The execution halts at main and is identified by ⇨.

Step 7: Choose Debug→Run or click the 🏃 (Run) toolbar button to run the program.

Step 8: Choose Debug→Halt to quit running the program.

2.5   Selecting a Project Configuration

A project configuration defines a set of project level build options. Options specified at this level apply to every file in the project.

Project configurations enable you to define build options for the different phases of program development. For example, you can define a Debug configuration to use while debugging your program and a Release configuration for building the finished product.

Each project is created with two default configurations: Debug and Release. Additional configurations can be defined. Whenever a project is created or an existing project is initially opened, the first configuration (in alphabetical order) is set active.

The active configuration setting is preserved in the CCS workspace.

When you build your program, the output files generated by the software tools are placed in a configuration-specific subdirectory. For example, if you have created a project in the directory MyProject, the output files for the Debug configuration are placed in MyProject\Debug. Similarly, the output files for the Release configuration are placed in MyProject\Release.

Change the Active Project Configuration

Click on the Select Active Configuration field in the Project toolbar and select a configuration from the drop-down list.

Figure 2–4. Change Active Project Configuration
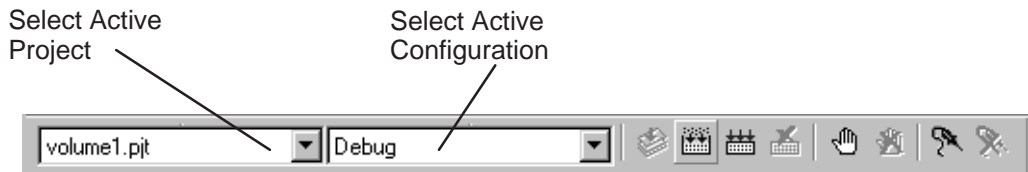
Add a New Project Configuration

Step 1: Select Project→Configurations, or right-click on the project's file-name in the Project View window and select Configurations.

Step 2: In the Project Configurations dialog box, click Add.

The Add Project Configuration window displays.
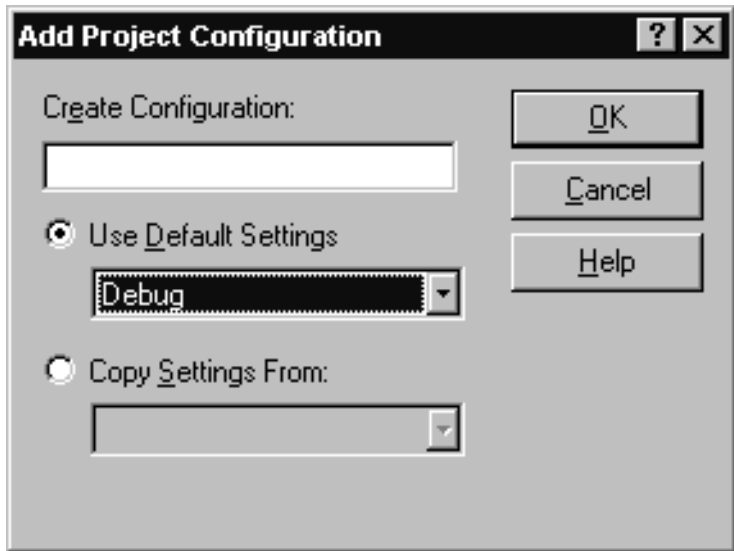
Step 3: In the Add Project Configuration dialog box, specify the name of the new configuration in the Create Configuration field, and choose to Use Default Settings (build options) or Copy Settings from an existing configuration to populate your new configuration.

Step 4: Click OK to accept your selections and exit the Add Project Configuration dialog.

Step 5: Click Close to exit the Project Configurations dialog.

Step 6: Modify your new configuration using the build options dialog found in the Project menu.

2.6 Building Projects From the Command Line

The timake.exe utility located in the <Installdir>\cc\bin directory provides a way to build Code Composer Studio projects outside of the main application. This utility can be used to accomplish batch builds.

2.7 Importing an External Makefile

The CCS IDE supports the use of external makefiles (*.mak) and an associated external make utility for project management and build process customization.

To enable the CCS IDE to build a program using a makefile, a CCS project must be created that wraps the makefile. After a CCS project is associated with the makefile, the project and its contents can be displayed in the Project View window and the Project→Build and Project→Rebuild All commands can be used to build the program.

Double-clicking on the name of the makefile in the Project View window opens the file for editing.

Special dialogs enable you to modify the makefile build commands and makefile options. The normal CCS Build Options dialogs are not available when working with makefiles.

Multiple configurations can be created, each with its own build commands and options.

Limitations and Restrictions

Source files can be added to or removed from the project in the Project View. However, changes made in the Project View do not change the contents of the makefile. These source files do not affect the build process nor are they reflected in the contents of the makefile. Similarly, editing the makefile does not change the contents in the Project View. File-specific options for source files that are added in the Project View are disabled. The Project→Compile File command is also disabled. However, when the project is saved, the current state of the Project View is preserved.

> **Note:**
>
> Before using CCS IDE commands to build your program using a makefile, it is necessary to set the necessary environment variables. To set environment variables, run the batch file
>
> DosRun.bat
>
> The batch file is located in the directory c:\ti. If you installed CCS IDE in a directory other than c:\ti, the batch file will be located in the directory you specified during installation.

2.8 Reviewing Your Source Code Using the Editor

Double-click on the *filename.*c file in the Project View to display the source code in the right half of the CCS window.

Figure 2–5. View Source Code
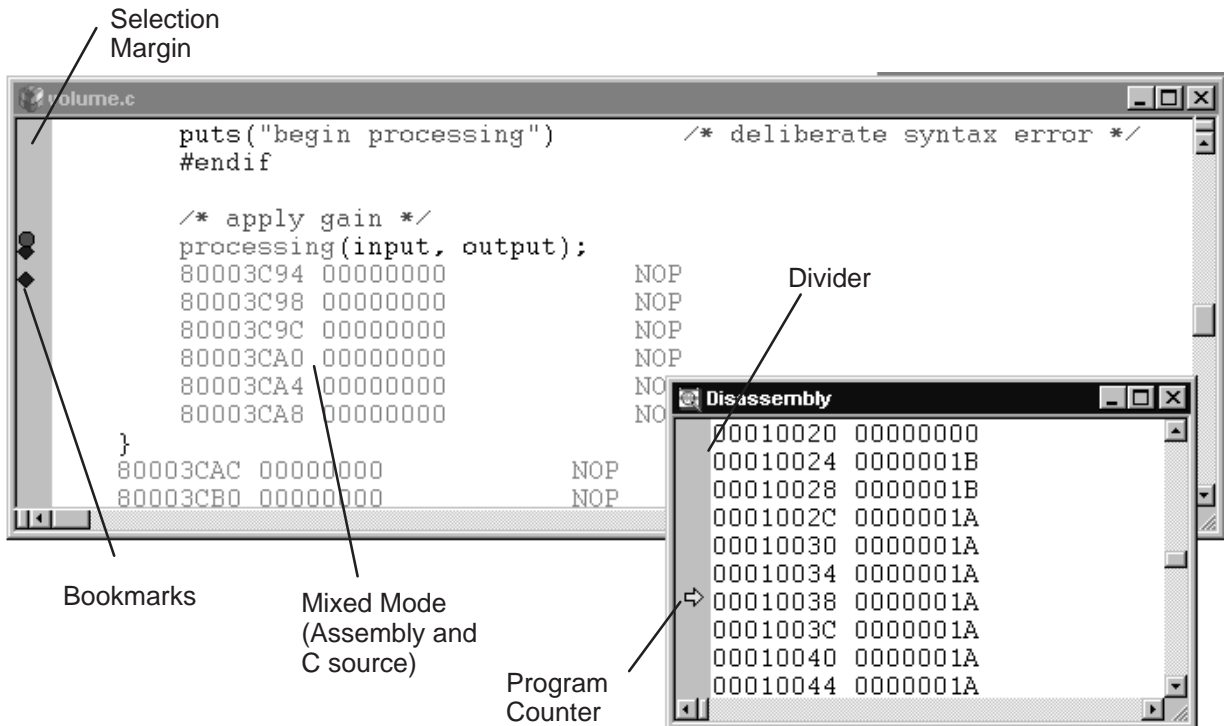
❑ **Selection Margin.** By default, a Selection Margin is displayed on the left-hand side of integrated editor and Disassembly windows. Colored icons in the Selection Margin indicate that a breakpoint (red) or Probe Point (blue) is set at this location. A yellow arrow identifies the location of the Program Counter (PC).

TIP: The Selection Margin can be resized by dragging the divider.

❑ **Keywords.** The integrated editor features keyword highlighting. Key-words, comments, strings, assembler directives, and GEL commands are highlighted in different colors.

TIP: In addition, new sets of keywords can be created, or the default keyword sets can be customized and saved in keyword files (*.kwd).

❑ **Keyboard Shortcuts.** The default keyboard shortcuts can be changed and new keyboard shortcuts can be created for any editing or debugging commands that can be invoked from a document window. Keyboard shortcuts can be modified through the customize dialog box in the Options menu.

❑ **Bookmarks.** Use bookmarks to find an maintain key locations within your source files. A bookmark can be set on any line of any source file.

CodeMaestro Settings

The CodeMaestro settings included in the CCS editor can help you be more productive.

Figure 2–6. CodeMaestro Settings Window
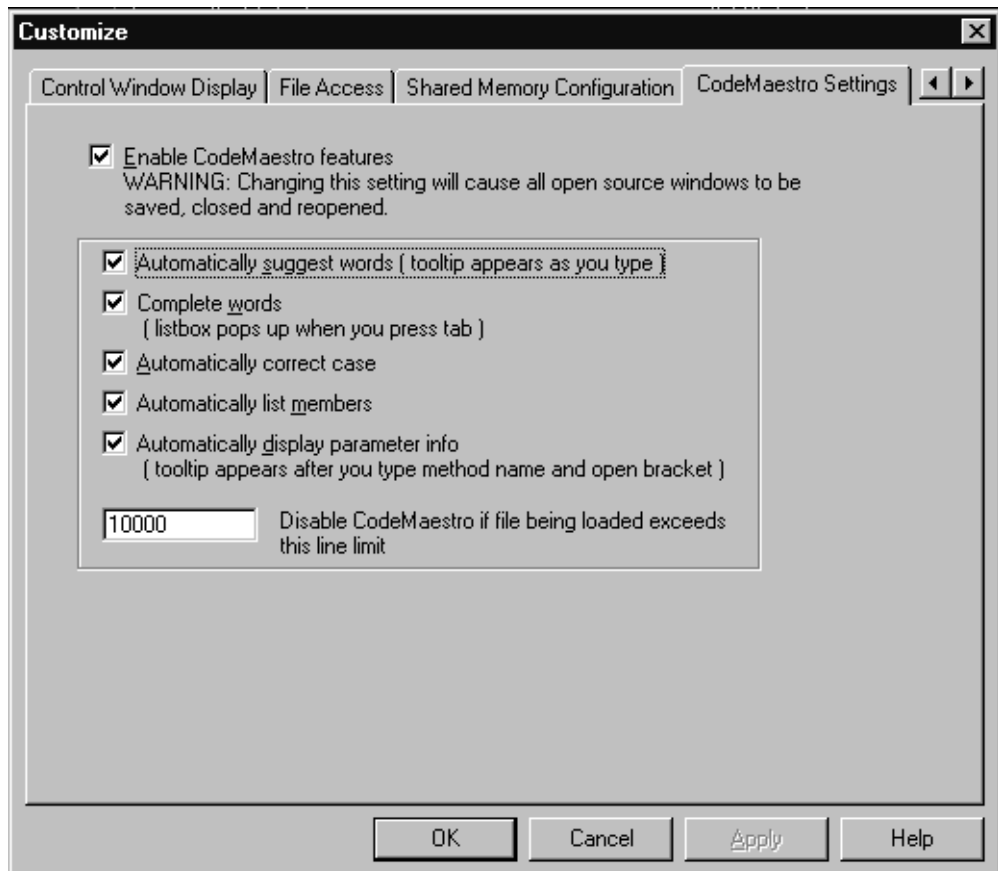
❑ **Automatically Suggest Word:** Helps you by suggesting a word or symbol that you have started typing.

❑ **Complete Word:** Allows you to select from a list of valid symbols.

❑ **Automatically correct case:** Corrects the case of your code.

❑ **Automatically List Members:** Displays a list containing all of the valid members of a structure or object.

❑ **Automatically Display Parameter Information:** Displays a tool-tip containing the parameter information for the function that you have started typing.

❑ **Line Limit:** Allows you to set a maximum number of lines per file, and if the file has more lines than the number you set, CodeMaestro is not invoked for that file.

External Editor

The CCS IDE supports the use of an external (third-party) text editor in place of the default integrated editor. When an external editor is configured and enabled, the external editor is launched whenever a new blank document is created or an existing file is opened. You can configure an external editor by selecting Options→Customize→Editor Properties.

An external editor can only be used to edit files. The CCS integrated editor must be used to debug your program.

Figure 2–7. External Editor Icon
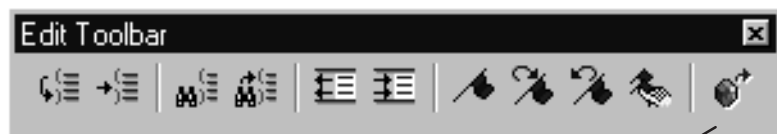
External Editor icon: toggle between an external editor and the CCS integrated editor

Code Composer Studio
Code Generation Tools

This chapter applies to all platforms using Code Composer Studio™
(CCS) IDE. Some of the Code Generation Tool sets discussed in this chapter may not be available for your ISA. For a complete listing of the tools available to you, see the online help and online documentation provided with the CCS IDE.

The different platforms offered with the CCS IDE are supported by a set of software development tools. These tools include an optimizing C/C++ compiler, an assembler, a linker, and assorted utilities. This chapter discusses these tools and shows you how to use them.

3.1 Code Generation Tools

Figure 3–1. Code Development Flow
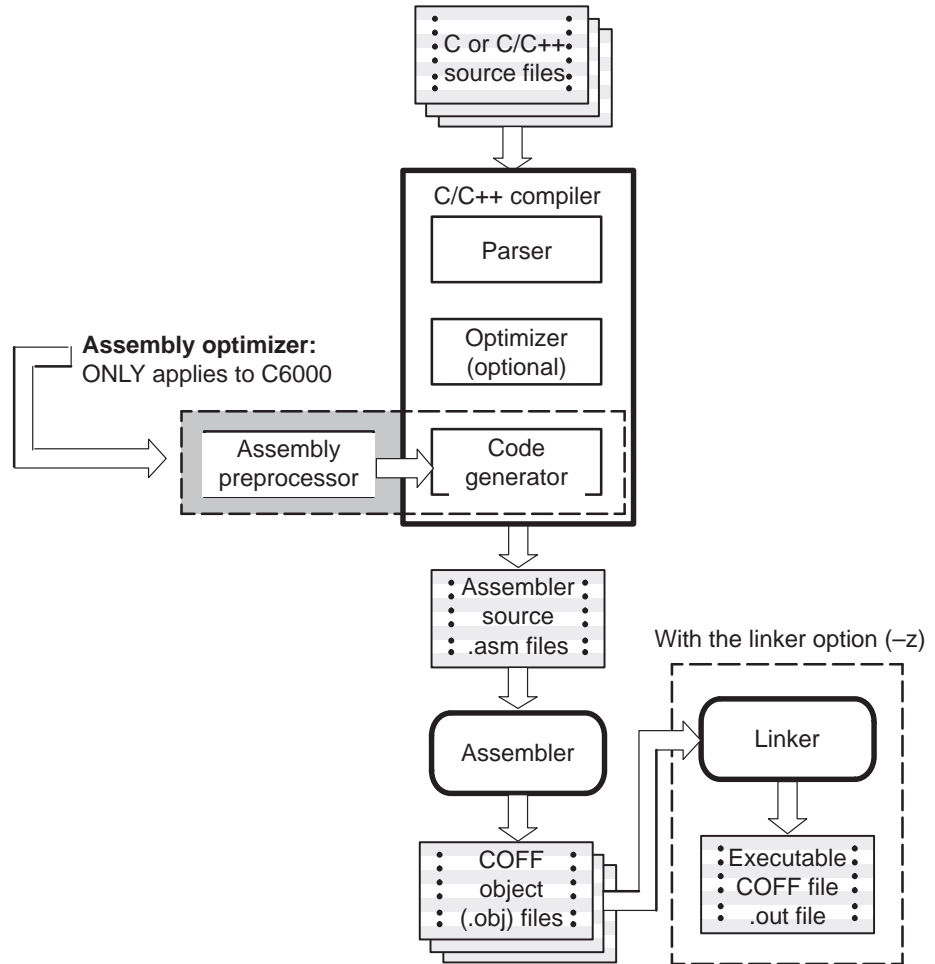
3.2 Code Generation Tools and Code Composer Studio

The CCS IDE provides a graphical interface for using the code generation tools.

A CCS project keeps track of all information needed to build a target program or library. A project records:

❑ Filenames of source code and object libraries
❑ Compiler, assembler, and linker options
❑ Include file dependencies

When you build a project with the CCS IDE, the appropriate code generation tools are invoked to compile, assemble, and/or link your program.

The compiler, assembler, and linker options can be specified within CCS's Build Options dialog box. Nearly all command line options are represented within this dialog box. Options that are not represented can be specified by typing the option directly into the editable text box that appears at the top of the dialog box.

Figure 3–2. Build Options Dialog Box
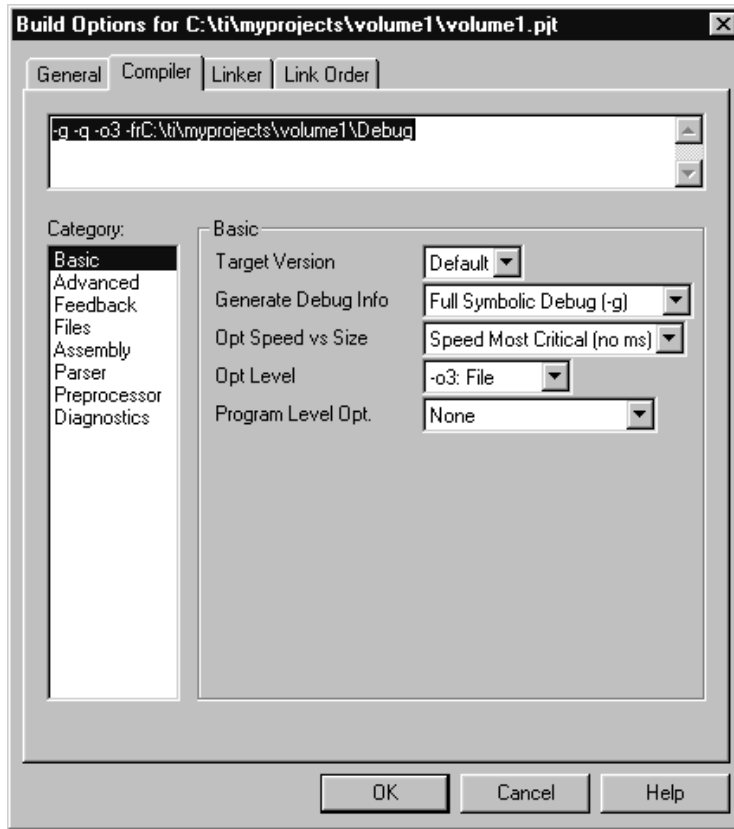
Build Options

You can set the compiler and linker options that are used when the CCS IDE builds your program.

Define a set of project level options that apply to all files in your project. Then, optimize your program by defining file-specific options for individual source code files.

TIP: For options that are commonly used together, you can set project level configurations, rather than have to set the same individual options repeatedly. For more information on setting project configurations, see section 2.5, Selecting a Project Configuration, on page 2-10. You can also look for this information in the online help and tutorial provided with the CCS IDE.

Set Project Level Options

Step 1: Select Project→Build Options.

Step 2: In the Build Options Dialog Box, select the appropriate tab.

Step 3: Select the options to be used when building your program.

Step 4: Click OK to accept your selections.

Set File-Specific Options

Step 1: Right-click on the name of the source file in the Project View window and select File Specific Options from the context menu.

Step 2: Select the options to be used when compiling this file.

Step 3: Click OK to accept your selections.

File-specific options are stored in the project file by recording only the differences between the project options and those set for the file.

3.3 Compiler Overview

The C and C++ compilers (for C5000™ and C6000™) are full-featured optimizing compilers that translate standard ANSI C programs into assembly language source. The following subsections describe the key features of the compilers.

Interfacing with Code Composer Studio

The following features pertain to interfacing with the compiler:

❑ **Compiler shell program**

The compiler tools include a shell program that you use to compile, assembly optimize, assemble, and link programs in a single step. For more information, see the *About the Shell Program* section in the *Optimizing Compiler User's Guide* appropriate for your device.

❑ **Flexible assembly language interface**

The compiler has straightforward calling conventions, so you can write assembly and C functions that call each other. For more information, see Chapter 8, *Run-Time Environment,* in the *Optimizing Compiler User's Guide* appropriate for your device.

3.4 Assembly Language Development Tools

The following is a list of the assembly language development tools:

❏ **Assembler.** The assembler translates assembly language source files into machine language object files. The machine language is based on common object file format (COFF).

❏ **Archiver.** The archiver allows you to collect a group of files into a single archive file called a *library*. Additionally, the archiver allows you to modify a library by deleting, replacing, extracting, or adding members. One of the most useful applications of the archiver is building a library of object modules.

❏ **Linker.** The linker combines object files into a single executable object module. As it creates the executable module, it performs relocation and resolves external references. The linker accepts relocatable COFF object files and object libraries as input.

❏ **Absolute Lister.** The absolute lister accepts linked object files as input and creates .abs files as output. You can assemble these .abs files to produce a listing that contains absolute, rather than relative, addresses. Without the absolute lister, producing such a listing would be tedious and would require many manual operations.

❏ **Cross-reference Lister.** The cross-reference lister uses object files to produce a cross-reference listing showing symbols, their definitions, and their references in the linked source files.

❏ **Hex-conversion Utility.** The hex-conversion utility converts a COFF object file into TI-Tagged, ASCII-hex, Intel, Motorola-S, or Tektronix object format. You can download the converted file to an EPROM programmer.

❏ With the **TMS320C54x** device, the **mnemonic-to-algebraic translator utility** converts assembly language source files. The utility accepts an assembly language source file containing mnemonic instructions. It converts the mnemonic instructions to algebraic instructions, producing an assembly language source file containing algebraic instructions.

3.5 Assembler Overview

The assembler translates assembly language source files into machine language object files. These files are in common object file format (COFF).

The two-pass assembler does the following:

❏ Processes the source statements in a text file to produce a relocatable object file

❏ Produces a source listing (if requested) and provides you with control over this listing

❏ Allows you to segment your code into sections and maintains a section program counter (SPC) for each section of object code

❏ Defines and references global symbols and appends a cross-reference listing to the source listing (if requested)

❏ Assembles conditional blocks

❏ Supports macros, allowing you to define macros inline or in a library

3.6 Linker Overview

The linker allows you to configure system memory by allocating output sections efficiently into the memory map. As the linker combines object files, it performs the following tasks:

❏ Allocates sections into the target system's configured memory

❏ Relocates symbols and sections to assign them to final addresses

❏ Resolves undefined external references between input files

The linker command language controls memory configuration, output section definition, and address binding. The language supports expression assignment and evaluation. You configure system memory by defining and creating a memory module that you design. Two powerful directives, MEMORY and SECTIONS, allow you to:

❏ Allocate sections into specific areas of memory

❏ Combine object file sections

❏ Define or redefine global symbols at link time

3.7 Visual Linker

There are two ways to link your code in the CCS IDE. One is textually using the linker command file and the other is graphically using the visual linker. The linker command file is a text file used explicitly by the standard linker to create the links automatically. The visual linker allows you to manually create these links in a graphical setting.

The Visual Linker is an interactive, extensible linker. Taking your application's object files/libraries and the target memory description as input, the Visual Linker provides a graphical means to configure system memory. You can use drag-and-drop manipulation to arrange the object files within a graphical representation of the memory layout. When you are satisfied with the memory layout, you can then generate the executable (.out) file.

Some features of the Visual Linker include:

❏ Visual drag-and-drop of program components into device-specific memory maps

❏ Immediate visual feedback on memory allocation to discover areas of optimization

❏ Reduced application size with elimination of unused code and data

❏ Wizards to ease migration from existing text linker control files

Getting Started with the Visual LInker

Before using the Visual Linker, go through the Visual Linker tutorial. In the CCS IDE, select Help→Tutorial. Choose the Visual Linker tutorial module.

Step 1: Select the Visual Linker as your project's linker.

Select Tools→Linker Configuration. In the Linker Configuration dialog box, select Use the Visual Linker and click OK.
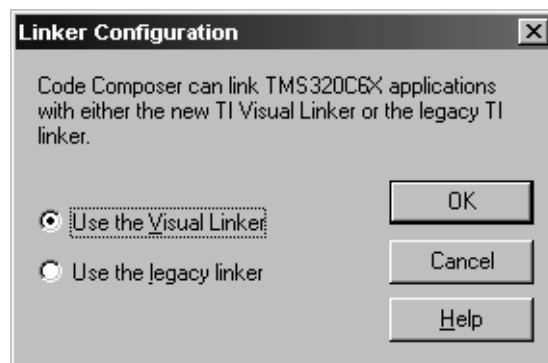
Step 2: Generate object files for your application before creating a recipe.

Open your CCS project. Select Project→Rebuild All. You will receive an error message in the CCS output window. Simply double-click on the error message to open the New Recipe wizard.

Step 3: Use the New Recipe wizard to create a recipe for the Visual Linker.

A recipe describes how to build an application. It is made up of:

■ **Ingredients**. The ingredients include a list of input files and a hardware description.

■ **Directions.** The directions come from a strategy for combining the ingredients. The strategy can be a file that was previously created, or it can be generated as you manipulate input files and other items within the Visual Linker.

New Recipe Wizard	☒

To accept the settings to the right, click the 'Accept' button. 〔 Accept 〕 Otherwise, click the 'Next' button below.	Recipe Name	vislink
	Recipe Location	C:\ti\myprojects\vislink
	Translate Sections Commands	Yes
	Output File	Vislink.out
	Map File	None
	Global Initialization	ROM
	Unused Code Elimination	None
	DSP/BIOS generated CMD file	Yes
	Full Link	

〔 < Back 〕 〔 Next > 〕 〔 Cancel 〕 〔 Help 〕
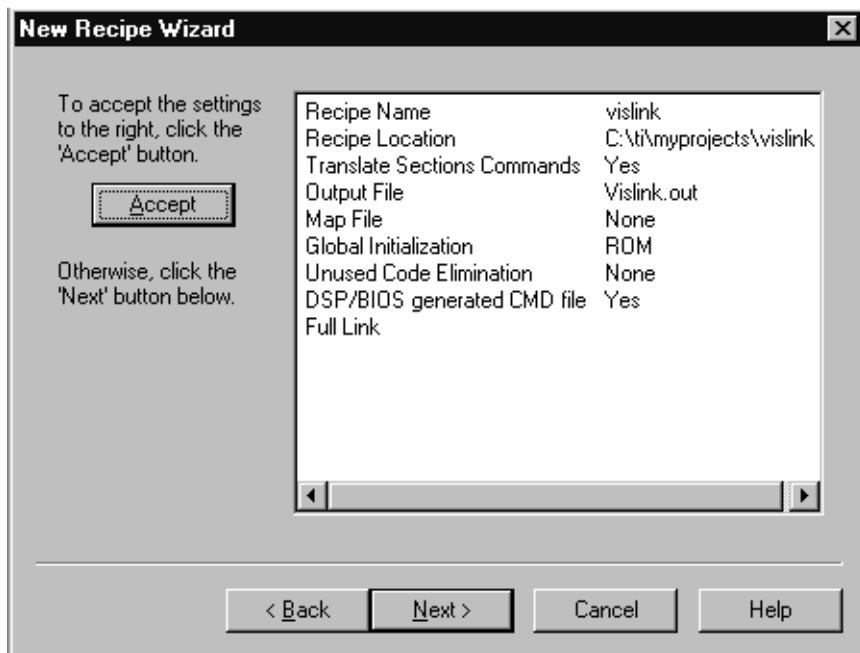
Step 4: Use the Visual Linker's views to arrange your program in memory.

The left pane of the Visual Linker is the tree view . The pane to the right of the tree view is the list view . The ingredients of the recipe appear as graphical elements within the Visual Linker's interface.

You can use drag-and-drop manipulation within these windows to arrange the components of your program in memory. After each drag

and drop manipulation, you see the effect immediately: the space used by each section, the space left over, etc.

Tree view   List View
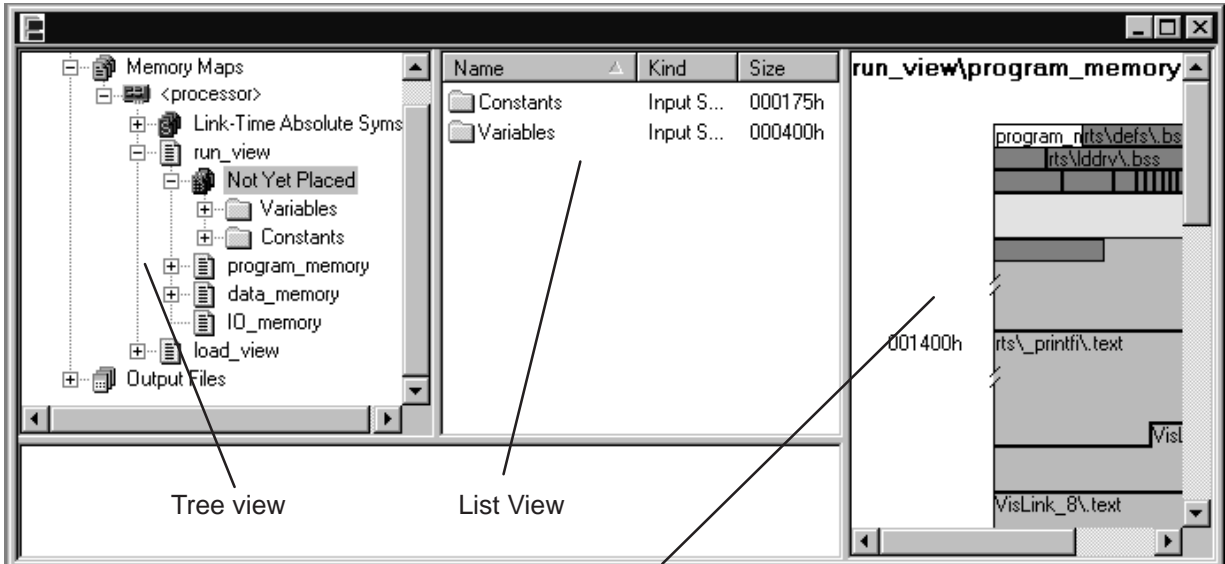
Graphical Elements
of the Recipe

Step 5: After you have arranged your program in memory, you can generate the executable file.

To generate an executable file, select Project→Rebuild All or click the ▦ (Rebuild All) toolbar button button on the CCS toolbar.

Step 6: You can then use the recipe when you run the Visual Linker from within a makefile or from the command line.

Note:

Use the vlnk command to run the Visual Linker from within a makefile or from the command line.

Some of the tasks that you will want to complete within the Visual Linker are:

❑ Placing input sections in memory
❑ Viewing portions of memory
❑ Specifying load and run addresses
❑ Avoiding allocation of reserved hardware addresses

3.8   C/C++ Code Development Tools

The following is a list of the C/C++ development tools:

❑   **C/C++   Compiler.** The C/C++ compiler accepts C/C++ source code and produces assembly language source code. A **shell program**, an **optimizer**, and an **interlist utility** are parts of the compiler.

■   The shell program enables you to compile, assemble, and link source modules in one step. If any input file has a .sa extension, the shell program invokes the assembly optimizer.

■   The optimizer modifies code to improve the efficiency of C programs.

■   The interlist utility interweaves C/C++ source statements with assembly language output.

❑   **Assembly Optimizer (C6000   only).** The assembly optimizer allows you to write linear assembly code without being concerned with the pipeline structure or with assigning registers. It accepts assembly code that has not been register-allocated and is unscheduled. The assembly optimizer assigns registers and uses loop optimization to turn linear assembly into highly parallel assembly that takes advantage of software pipelining.

❑   **Library-build Utility.** You can use the library-build utility to build your own customized run-time-support library. Standard run-time-support library functions are provided as source code in rts.src and rstcpp.src. The object code for the run-time-support functions is compiled for little-endian mode versus big-endian mode and C code versus C++ code into standard libraries.

The **run-time-support libraries** contain the ANSI standard run-time-support functions, compiler-utility functions, floating-point arithmetic functions, and C I/O functions that are supported by the compiler.

❑   **C++ Name Demangling Utility.** The C++ compiler implements function overloading, operator overloading, and type-safe linking by encoding a function's signature in its link-level name. The process of encoding the signature into the linkname is often referred to as name mangling. When you inspect mangled names, such as in assembly files or linker output, it can be difficult to associate a mangled name with its corresponding name in the C++ source code. The C++ name demangler is a debugging aid that translates each mangled name it detects to its original name found in the C++ source code.

The following is a list of available products for refining and correcting your code:

❑ **TMS320C6000** devices:

   ■ An instruction-accurate and clock-accurate software simulator
   ■ An extended development system (XDS510™) emulator

❑ **TMS320C54x** devices:

   ■ An instruction-accurate software simulator
   ■ An extended development system (XDS510™) emulator
   ■ An evaluation module (EVM)

❑ **TMS320C55x** devices:

   ■ An instruction-accurate software simulator
   ■ An extended development system (XDS510™) emulator

These tools are accessed within the CCS IDE. For more information, see the online help provided with the CCS IDE.

Code Composer Studio Debug Tools

This chapter applies to all platforms using Code Composer Studio™ (CCS) IDE. *However, not all devices have access to all of the tools discussed in this chapter. For a complete listing of the tools available to you, see the on-line help and online documentation provided with the CCS IDE.*

The CCS IDE comes with a number of tools that help you debug your programs. This chapter discusses these tools and shows you how to use them.

4.1 Overview of Applicable Debug Tools

The following table shows the debug tools discussed in this chapter that are used by different generations of devices. However, the specific devices that fall in each generation may or may not have access to these debug tools. To see a complete list of debug tools available for your device, access the online help provided with the CCS IDE.

Table 4–1. Debug Tools

Platform/Device	Breakpoints	Watch Window	Probe Points	Simulator Analysis	Emulator Analysis	Event Triggering	Graphs	Symbol Browser	Command Window	Pin connect	Port Connect	Data Converter
TMS320C62x	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
TMS320C64x	✓	✓	✓				✓	✓	✓			
TMS320C67x	✓	✓	✓	✓	✓	✓	✓	✓	✓		✓	✓
TMS320C54x	✓	✓	✓	✓	✓		✓	✓	✓	✓		✓
TMS320C55x	✓	✓	✓		✓		✓	✓	✓	✓	✓	

4.2 Introduction to Breakpoints

Breakpoints are an essential component of any debugging session.

Breakpoints stop the execution of the program. While the program is stopped, you can examine the state of the program, examine or modify variables, examine the call stack, etc. Breakpoints can be set on a line of source code in an Editor window or a disassembled instruction in the Disassembly window. After a breakpoint is set, it can be enabled or disabled.

For breakpoints set on source lines it is necessary that there be an associated line of dissassembly code. When compiler optimization is turned on, many source lines cannot have breakpoints set. To see allowable lines, use mixed mode in the editor window.

Note:

CCS IDE tries to relocate a breakpoint to a valid line in your source window. CCS IDE places a breakpoint icon in the selection margin beside the lineon which it locates the breakpoint. If CCS IDE cannot determine an allowable line, it reports an error in the message window.

Breakpoints are saved in the project workspace. For information on analyzing breakpoints, see section 4.5, Simulator Analysis, on page 4-14, and section 4.6, Emulator Analysis, on page 4-16.

Note:

Composer Studio briefly halts the target whenever it reaches a Breakpoint or Probe Point. Therefore, the target application may not meet real-time deadlines if you are using Probe Points. At this stage of development, you are testing the algorithm. Later, you can analyze real-time behavior using RTDX and DSP/BIOS.

For real-time debugging information, see Chapter 6, Code Composer Studio Real Time Components.

Software Breakpoints

Breakpoints can be set in any Disassembly window or document window containing C/C++ source code. There is no limit to the number of software breakpoints that can be set, provided they are set at writable memory locations (RAM). Software breakpoints operate by modifying the target program to add a breakpoint instruction at the desired location.

The fastest way to set a breakpoint is to simply double-click on the desired line of code.

Step 1: In a document window or Disassembly window, move the cursor over the line where you want to place a breakpoint.

Step 2: In a document window, double-click in the Selection Margin immediately preceding the line.

In a Disassembly window, double–click on the desired line.

A breakpoint icon in the Selection Margin indicates that a breakpoint has been set at the desired location.

The Toggle Breakpoint command and the Toggle Breakpoint button also enable you to quickly set and clear breakpoints.

Step 1: In a document window or Disassembly window, put the cursor in the line where you want to set the breakpoint.

Step 2: Right-click and select Toggle Breakpoint, or click the Toggle Breakpoint button on the Project toolbar.

Toggle Breakpoint:

Hardware Breakpoints

Hardware breakpoints differ from software breakpoints in that they do not modify the target program; they use hardware resources available on the chip. Hardware breakpoints are useful for setting breakpoints in ROM memory or breaking on memory accesses instead of instruction acquisitions. A breakpoint can be set for a particular memory read, memory write, or memory read or write. Memory access breakpoints are not shown in the source or memory windows.

Hardware breakpoints can also have a count, which determines the number of times a location is encountered before a breakpoint is generated. If the count is 1, a breakpoint is generated every time.

Hardware breakpoints cannot be implemented on a simulated target.

To set a hardware breakpoint:

Step 1: Select Debug→Breakpoints. The Break/Probe Points dialog box appears with the Breakpoints tab selected.

Step 2: In the Breakpoint type field, choose "H/W Break at location" for instruction acquisition breakpoints or choose "Break on <bus> <Read|Write|R/W>" at location for a memory access breakpoint.

Step 3: Enter the program or memory location where you want to set the breakpoint. Use one of the following methods:

- For an absolute address, you can enter any valid C expression, the name of a C function, or a symbol name.

- Enter a breakpoint location based on your C source file. This is convenient when you do not know where the C instruction is in the executable. The format for entering in a location based on the C source file is as follows: fileName line lineNumber.

Step 4: Enter the number of times the location is hit before a breakpoint is generated, in the Count field. Set the count to 1 if you wish to break every time.

Step 5: Click the Add button to create a new breakpoint. This causes a new breakpoint to be created and enabled.

Step 6: Click OK.

4.3 Watch Window

When debugging a program, it is often helpful to understand how the value of a variable changes during program execution. The Watch window allows you to monitor the values of local and global variables and C/C++ expressions.

To open the Watch window:

Select View→Watch Window, or click the Watch Window button on the Watch toolbar.

Figure 4–1. Watch Window Toolbar


Open Quick Watch
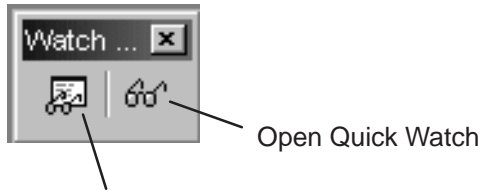
Open the Watch window

The Watch window contains two tabs labeled: Watch Locals and Watch.

❑ In the Watch Locals tab, the debugger automatically displays the Name, Value, and Type of the variables that are local to the currently executing function.

❑ In the Watch tab, the debugger displays the Name, Value, and Type of the local and global variables and expressions that you specify.

Figure 4–2. Watch Window
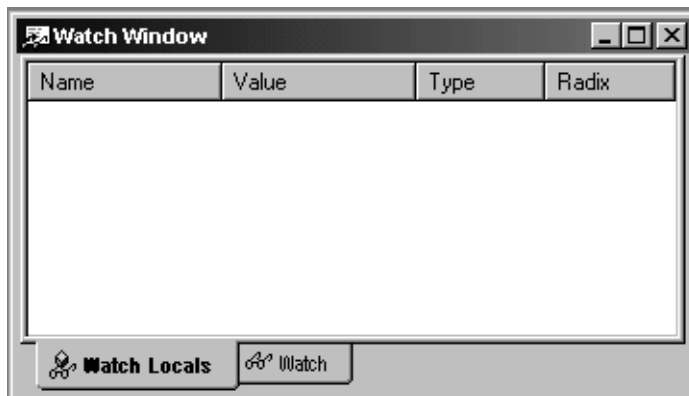
For detailed information on the Watch Window, see the Watch Window topics provided in the online help: Help→Contents→Watch Window.

When you are developing and testing programs, you often need to check the value of a variable during program execution. In this section, you use break-points and the Watch Window to view such values. You also use the step commands after reaching the breakpoint.

Step 1: Choose File→Load Program.

Step 2: Double–click on the *filename*.c file in the Project View.

Step 3: Put your cursor in a line that allows breakpoints.

Step 4: Click the 🖑 (Toggle Breakpoint) toolbar button or press F9.

The selection margin indicates that a breakpoint has been set (red icon). If you disable the selection margin (Options→Customize) the line is highlighted in magenta.
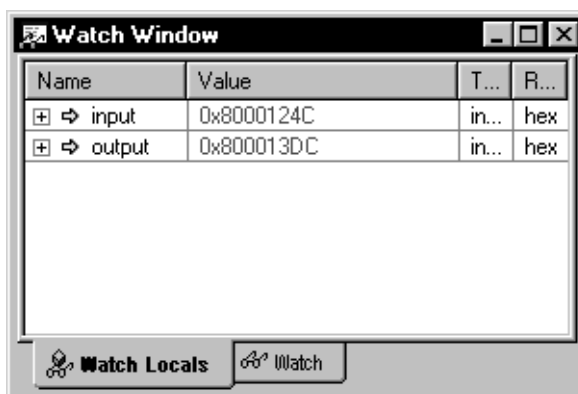
Step 5: Choose View→Watch Window.

A separate area in the lower–right corner of the CCS window appears. At run time, this area shows the values of watched variables.

By default, the Locals tab is selected and displays Local variables that are local to the function being executed.

Step 6: If not at main, choose Debug→Go Main.

Step 7: Choose Debug→Run, or press F5, or press the 🏃 Icon.

Name	Value	T...	R...
⊞ ⇨ input	0x8000124C	in...	hex
⊞ ⇨ output	0x800013DC	in...	hex

🐾 **Watch Locals**   👓 Watch

Step 8: Select the Watch tab.

Step 9: Click on the expression icon 📋 in the Name column and type the name of the variable to watch.

Step 10: Click on the white space in the watch window to save the change. The value should immediately appear, similar to this example.
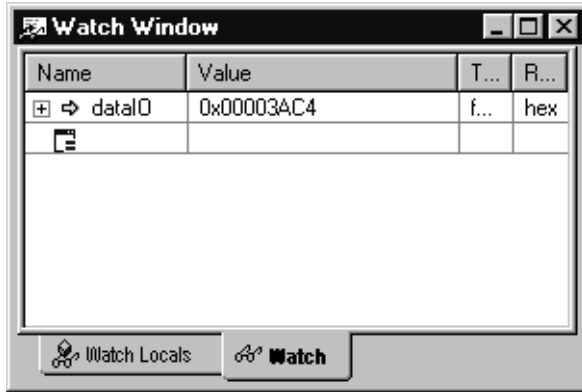
Step 11: Click the (Step Over) toolbar button or press F10 to step over the call to your watched variable.

Other commands you can use are:

 Step Into (F8)

 Step Over (F10)

 Step Out (Shift F7)

 Run to Cursor (Ctrl F10)

In addition to watching the value of a simple variable, you can watch the values of the elements of a structure.
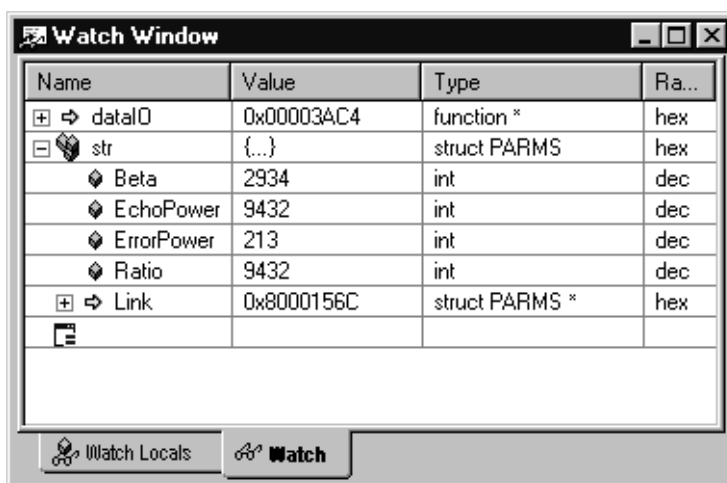
Step 1: Select the Watch tab.

Step 2: Click on the expression icon in the Name column and type the name of the expression to watch.

Step 3: Click on the white space in the watch window to save the change.

Step 4: Click once on the + sign. The CCS IDE expands this line to list all the elements of the structure and their values. (The address shown for Link may vary.)

Name	Value	Type	Ra...
⊞ ⇨ dataI0	0x00003AC4	function *	hex
⊟ 🏵 str	{...}	struct PARMS	hex
◆ Beta	2934	int	dec
◆ EchoPower	9432	int	dec
◆ ErrorPower	213	int	dec
◆ Ratio	9432	int	dec
⊞ ⇨ Link	0x8000156C	struct PARMS *	hex
⬚			

Watch Locals   **Watch**

Step 5: Double-click on the Value of any element in the structure to edit the value for that element.

Step 6: Change the value of a variable.

Notice that the value changes in the Watch Window. The value also changes color to red, indicating that you have changed it manually.

4.4   Probe Points

In this section, you add a Probe Point, which reads data from a file on your PC. Probe Points are a useful tool for algorithm development. You can use probe points to:

❏ transfer input data from a file on the host PC to a buffer on the target for use by the algorithm.

❏ transfer output data from a buffer on the target to a file on the host PC for analysis.

❏ update a window, such as a graph, with data.

More Information About Probe Points

Probe Points are similar to breakpoints in that they both halt the target to perform their action. However, Probe Points differ from breakpoints in the following ways:

❏ Probe Points halt the target momentarily, perform a single action, and resume target execution.

❏ Breakpoints halt the CPU until execution is manually resumed and cause all open windows to be updated.

❏ Probe Points permit automatic file input or output to be performed; breakpoints do not.

This section shows how to use a Probe Point to transfer the contents of a PC file to the target for use as test data. It also uses a breakpoint to update all the open windows when the Probe Point is reached.

Step 1: Choose File→Load Program. Select *filename*.out, and click Open.

Step 2: Double-click on the *filename*.c file in the Project View.

Step 3: Put your cursor in a line of the main function to which you want to add a probe point.

Step 4: Click the ▧ (Toggle Probe Point) toolbar button.

Step 5: From the File menu, choose File I/O. The File I/O dialog appears so that you can select input and output files.
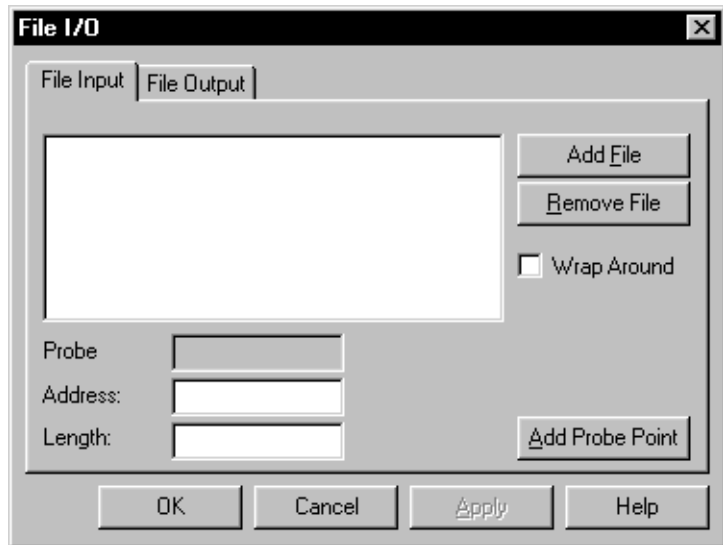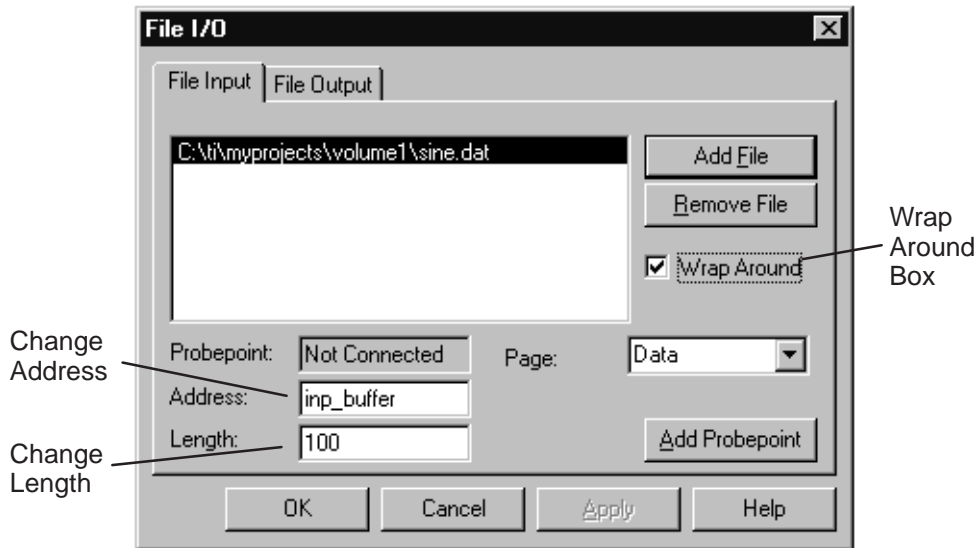

Step 6: In the File Input tab, click Add File.

Step 7: Browse to your project folder, select *filename*.dat and click Open.

A control window for the *filename*.dat file appears. When you run the program, you can use this window to start, stop, rewind, or fast forward within the data file.
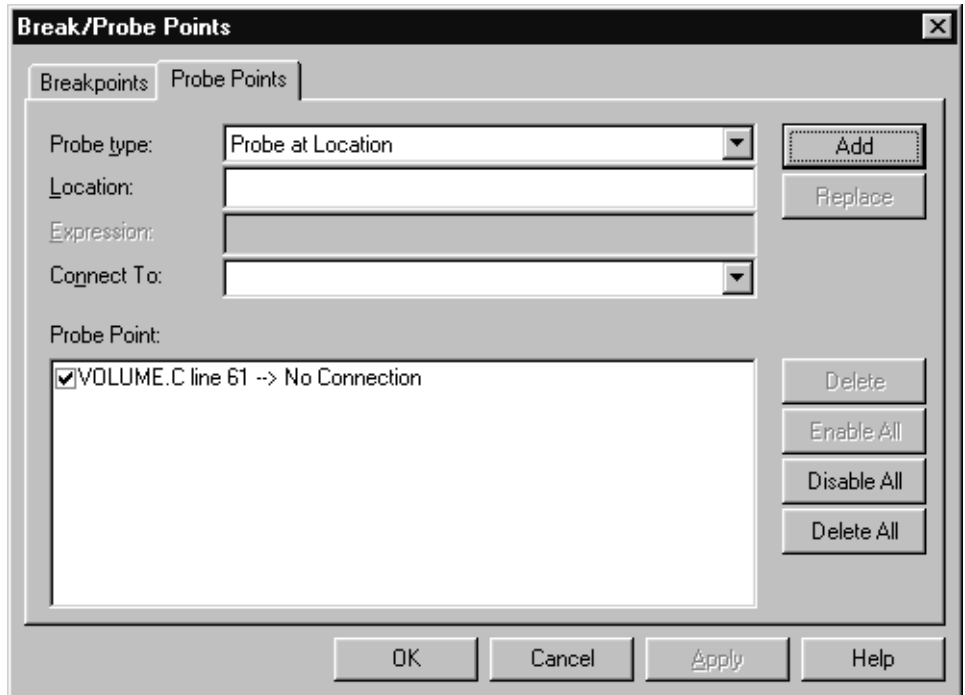
Step 8: In the File I/O dialog, change the Address and the Length values. Also, put a check mark in the Wrap Around box.

Change
Address

Change
Length

Wrap
Around
Box


```
File I/O                                              ✕

  File Input │ File Output │

  C:\ti\myprojects\volume1\sine.dat          Add File
                                             Remove File
                                           ☑ Wrap Around

  Probepoint:  Not Connected    Page:    Data        ▼
  Address:     inp_buffer
  Length:      100                         Add Probepoint

        OK          Cancel        Apply         Help
```

- The Address field specifies where the data from the file is to be placed.

- The Length field specifies how many samples from the data file are read each time the Probe Point is reached.

- The Wrap Around option causes the CCS IDE to start reading from the beginning of the file when it reaches the end of the file. This allows the data file to be treated as a continuous stream of data.

Step 9: Click Add Probe Point. The Probe Points tab of the Break/Probe Points dialog appears.

Break/Probe Points   ☒

   Breakpoints   Probe Points

   Probe type: Probe at Location   ▼   [Add   ]

   Location: [   ]   [Replace]

   Expression: [   ]

   Connect To: [▼]

   Probe Point:

   ☑VOLUME.C line 61 --> No Connection   [Delete]

                                                                  [Enable All]

                                                                  [Disable All]

                                                                  [Delete All]

              [OK   ]   [Cancel]   [Apply]   [Help]

Step 10: In the Probe Point list, highlight a line.

Step 11: In the Connect To field, click the down arrow and select a file from the list.

Step 12: Click Replace. The Probe Point list changes to show that this Probe Point is connected to the sine.dat file.

Step 13: Click OK. The File I/O dialog shows that the file is now connected to a Probe Point.

Step 14: Click OK to close the File I/O dialog.

4.5 Simulator Analysis

The Simulator Analysis tool reports the occurrence of particular system events so you can accurately monitor and measure the performance of your program.
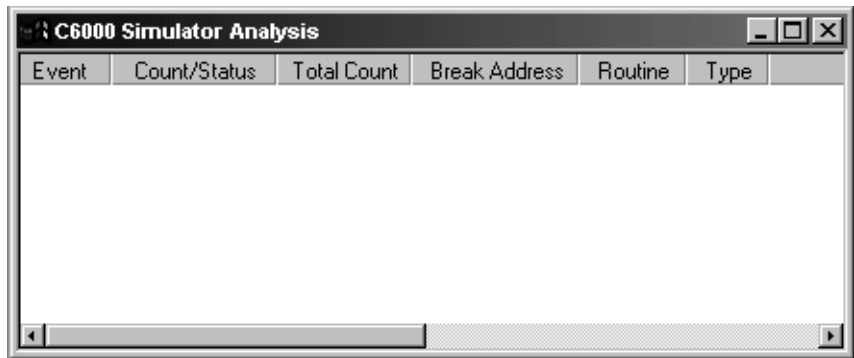
User Options:

❑ Enable/disable analysis
❑ Count the occurrence of selected events
❑ Halt execution whenever a selected event occurs
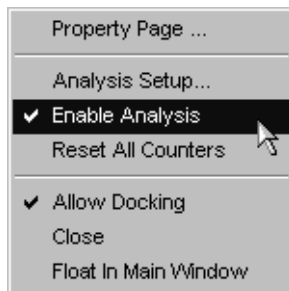❑ Delete count or break events
❑ Create a log file
❑ Reset event counter

To use the Simulator Analysis tool:

Step 1: Load your program using the CCS IDE.

Step 2: Start the analysis tool. Select Tools→Simulator Analysis for your device.

Step 3: Right-click in the Simulator Analysis window and select Enable analysis.

Step 4: Specify your analysis parameters (count events or break events).

Step 5: Use the CCS IDE to run or step through your program.
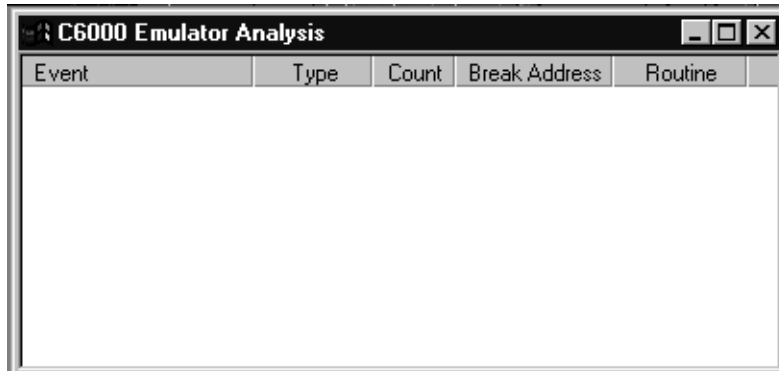
Step 6: Analyze the output of the analysis tool.

For detailed information on the Simulator Analysis tool, see the Simulator Analysis topics provided in the online help: Help→Contents→Simulator Analysis.

4.6 Emulator Analysis

The Emulator Analysis tool allows you to set up, monitor, and count events and hardware breakpoints.

Figure 4–3. Emulator Analysis Window

To start the Emulator Analysis tool:

Step 1: Load your program using the CCS IDE.

Step 2: Select Tools→Emulator Analysis for your device from the CCS menu bar.

The Emulator Analysis window contains the following information:

This column. . .	displays. . .
Event	the event name.
Type	whether the event is a break or count event.
Count	the number of times the event occurred before the program halted.
Break Address	the address at which the break event occurred.
Routine	the routine in which the break event occurred.

Note:

You cannot use the analysis features while you are using the profiling clock.

For detailed information on the Emulator Analysis tool, see the Emulator Analysis topics provided in the online help: Help→Contents→Emulator Analysis.

4.7 Advanced Event Triggering

Advanced Event Triggering (AET) is supported by a group of tools that makes hardware analysis easier than before. Advanced Event Triggering uses Event Analysis and State Sequencer to simplify hardware analysis.

Event Analysis uses a simple interface to help you configure common hardware debug tasks called jobs. Setting breakpoints, action points, and counters is easy using a right-click menu and simple drag-and-drop. You can access Event Analysis from the tools menu, or by right-clicking in a source file.

Event Sequencer allows you to look for conditions that you specify in your target program and initiates specific actions when these conditions are detected. While the CPU is halted, you define the conditions and actions, then run your target program. The sequence program then looks for the condition you specified and performs the action you requested.

Event Analysis

The following jobs can be performed using Event Analysis:

❑ Setting Breakpoints
  ■ Hardware Breakpoint
  ■ Hardware Breakpoint With Count
  ■ Chained Breakpoint
  ■ Global Hardware Breakpoint

❑ Setting Action/Watch Points
  ■ Data Actionpoint
  ■ Program Actionpoint
  ■ Watchpoint
  ■ Watchpoint With Data

❑ Setting Counters
  ■ Data Access Counter
  ■ Profile Counter
  ■ Watchdog Timer
  ■ Generic Counter

❑ Other
  ■ Benchmark to Here
  ■ Emulation Pin Configuration

For detailed information on the Event Analysis tool, see the Event Analysis topics provided in the online help: Help→Contents→Advanced Event Triggering.

To configure a job using the Event Analysis Tool, CCS IDE must be configured for a target processor that contains on-chip analysis features. You can use Event Analysis by selecting it from the Tools menu or by right-clicking in a source file. Once you configure a job, it is enabled and will perform analysis when you run code on your target. For information about how to enable or disable a job that is already configured, see the Advance Event Triggering online help.
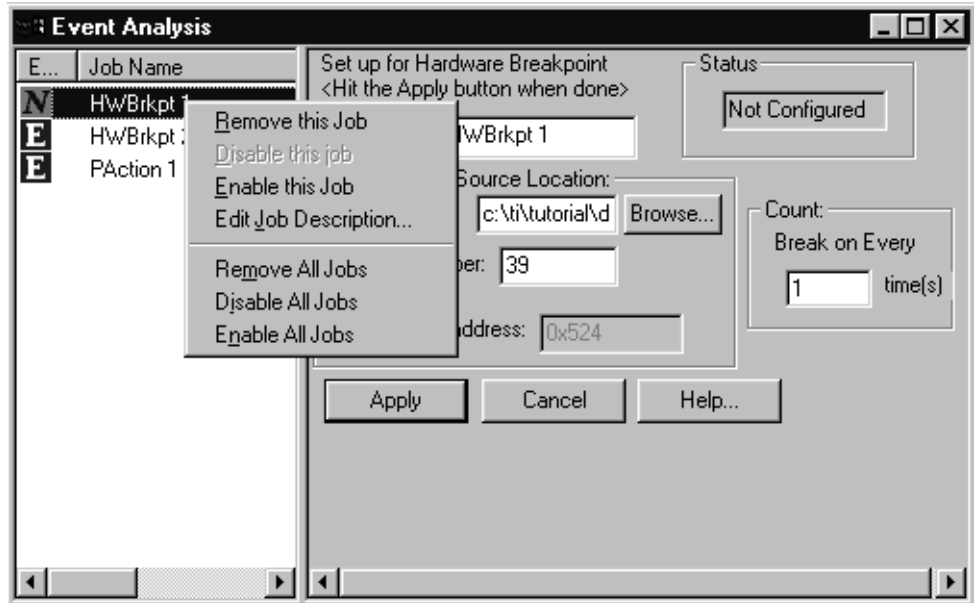
Step 1: Select Tools→Advanced Event Triggering→Event Analysis.

The Event Analysis window displays.
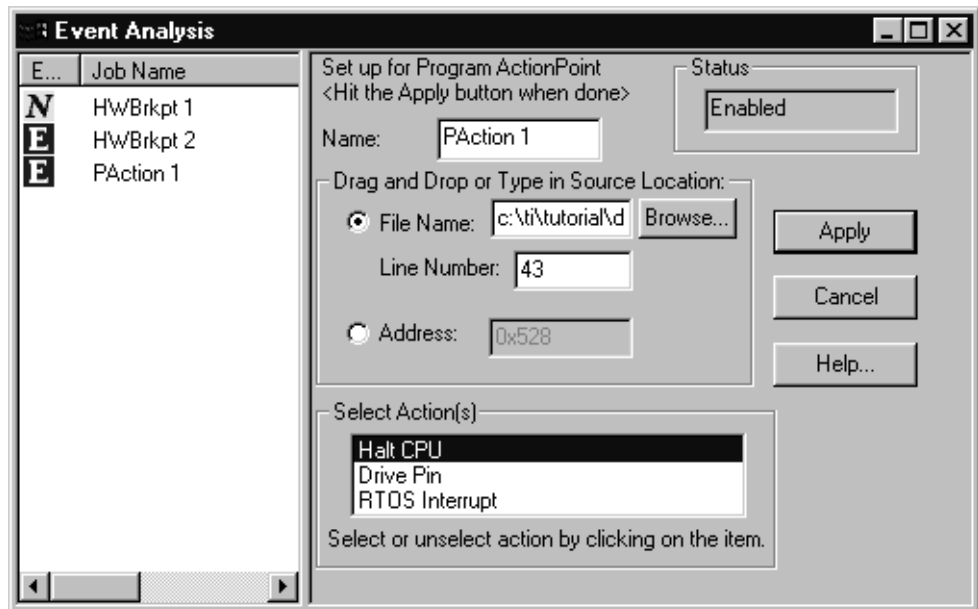
Step 2: Right-click in the Event Analysis Window and choose Event Triggering→Job Type→Job.

The job menu is dynamically built and dependent on the target configuration. If a job is not supported on your target, the job is grayed out.

Step 3: Type your information in the Job dialog box.

**Step 4:** Click Apply to program the job and save your changes.

Event Sequencer

The Event Sequencer allows you to look for conditions that you specify in your target program and initiates specific actions when these conditions are detected. While the CPU is halted, you define the actions, then run your target program. The sequencer program then looks for the condition that you specified and performs the action you requested.

To use the Event Sequencer, CCS IDE must be configured for a target processor that contains on-chip analysis features. You can use the Event Sequencer by selecting it from the Tools menu. Once you create an Event Sequencer program, it is enabled and performs analysis when you run code on your target. For information on creating an Event Sequencer program, see the Advanced Event Triggering online help.

To enable the Event Sequencer:

Step 1: Select Tools→Advanced Event Triggering→Event Sequencer.

The Event Sequencer window displays.
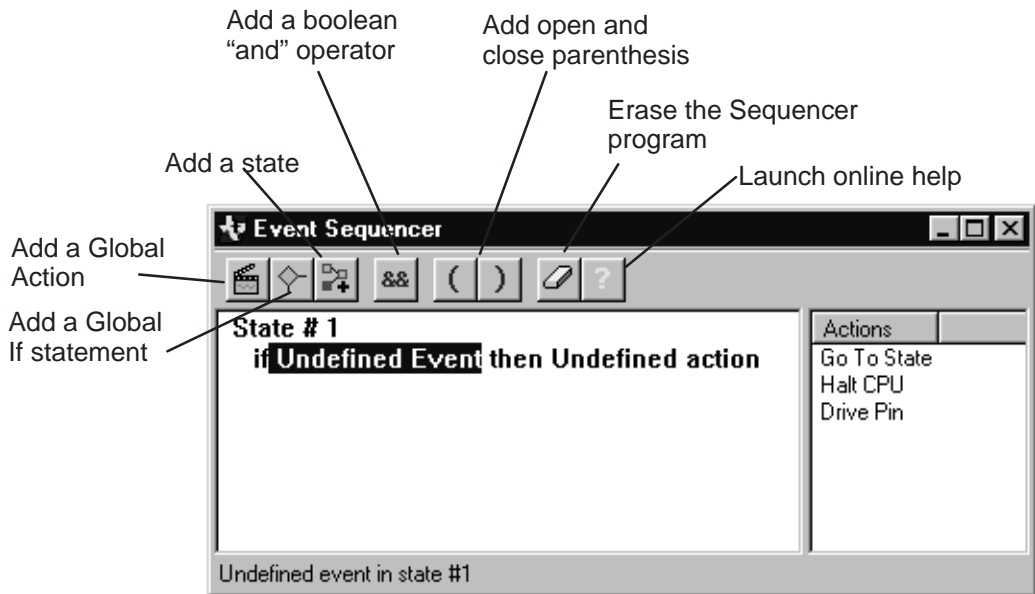
Step 2: Right-click in the Event Sequencer window or use the Event Sequencer toolbar buttons to create a sequencer program.
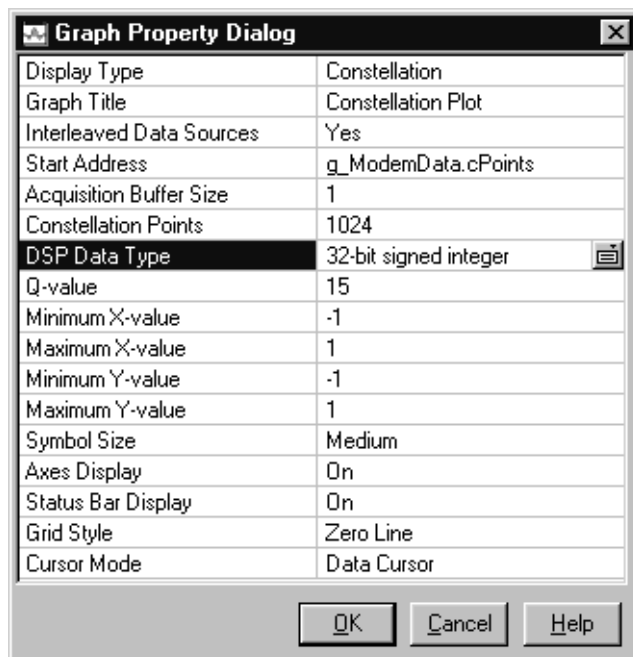
4.8 Displaying Graphs

If you ran a program using only breakpoints and watch windows, you would not see much information about what the program was doing. You could set watch variables on addresses within the inp_buffer and out_buffer arrays, but you would need to watch a lot of variables and the display would be numeric rather than visual.

CCS IDE provides a variety of ways to graph data processed by your program. In this example, you view a signal plotted against time.

Step 1: Choose View→Graph→Time/Frequency.

Step 2: In the Graph Property Dialog, change the Graph Title, Start Address, Acquisition Buffer Size, Display Data Size, DSP Data Type, Autoscale, and Maximum Y-value properties to the values shown here. Scroll down or resize the dialog box to see all the properties.

Graph Property Dialog	
Display Type	Constellation
Graph Title	Constellation Plot
Interleaved Data Sources	Yes
Start Address	g_ModemData.cPoints
Acquisition Buffer Size	1
Constellation Points	1024
DSP Data Type	32-bit signed integer
Q-value	15
Minimum X-value	-1
Maximum X-value	1
Minimum Y-value	-1
Maximum Y-value	1
Symbol Size	Medium
Axes Display	On
Status Bar Display	On
Grid Style	Zero Line
Cursor Mode	Data Cursor

OK   Cancel   Help

Step 3: Click OK. A graph window for the Input Buffer appears.

Step 4: Right-click on the Input window and choose Clear Display from the pop-up menu.

Step 5: Choose View→Graph→Time/Frequency again.

Step 6: This time, change the Graph Title to Output and the Start Address to out_buffer. All the other settings are correct.

Step 7: Click OK to display the graph window for the Output. Right-click on the graph window and choose Clear Display from the pop-up menu.

4.9 Symbol Browser

The Symbol Browser displays all of the associated files, functions, global variables, types, and labels of a loaded COFF output file (*.out). The Symbol Browser contains five tabbed windows:

❑ Files
❑ Functions
❑ Globals
❑ Types
❑ Labels

Each tabbed window contains nodes representing various symbols. A plus sign (+) preceding a node indicates that the node can be further expanded. To expand the node, simply click the + sign. A minus sign (–) precedes an expanded node. Click the – sign to hide the contents of that node.

To open the Symbol Browser, select Tools→Symbol Browser.

Figure 4–4. Symbol Browser Window

For detailed information on the Symbol Browser tool, see the Symbol Browser topics provided in the online help: Help→Contents→Symbol Browser.

4.10 General Extension Language (GEL)

The General Extension Language (GEL) is an interpretive language, similar to C, that lets you create functions to extend CCS IDE's usefulness.

You create your GEL functions using the GEL grammar, and then load them into CCS IDE. With GEL, you can access actual/simulated target memory locations and add options to CCS IDE's GEL menu. GEL is particularly useful for automated testing and user workspace customization.

You can call GEL functions from anywhere that you can enter an expression. You can also add GEL functions to the Watch window so they execute at every breakpoint.

CCS IDE comes equipped with many built-in GEL functions to help you automate common debugging tasks.

4.11 Command Window

The Command Window enables you to specify commands to the CCS debugger using the TI Debugger command syntax.

Many of the commands accept C expressions as parameters. This allows the instruction set to be relatively small, yet powerful. Because C expressions can have side effects (that is, the evaluation of some types of expressions can affect existing values) you can use the same command to display or change a value.

To open the Command Window:

Select Tools→Command Window from the CCS menu bar.

Figure 4–5. Command Window

| Output window | Command field | Execution (or halt) button |
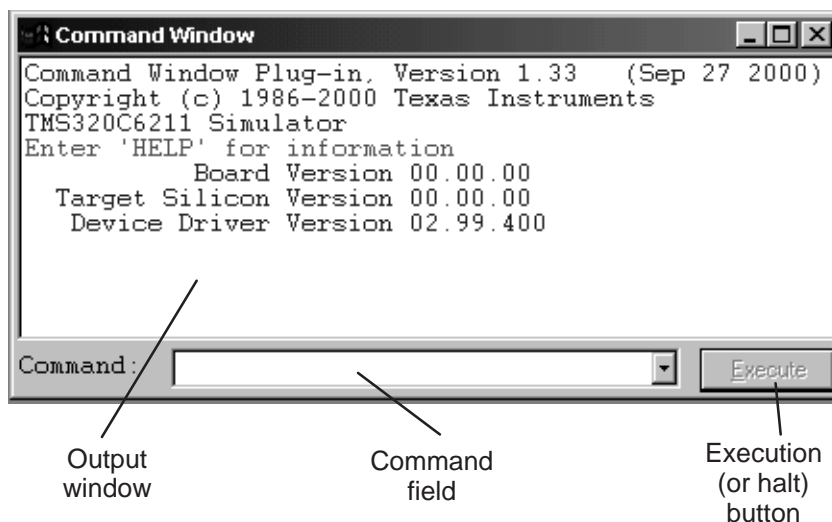
For detailed information on the Command Window, see the Command Window topics provided in the online help: Help→Contents→Command Window.

4.12 Pin Connect

The Pin Connect tool enables you to specify the interval at which selected external interrupts occur.

To simulate external interrupts:

Step 1: Create a data file that specifies interrupt intervals.

Step 2: Start the Pin Connect tool. From the Tools menu, choose Pin Connect.

Pin name

File name
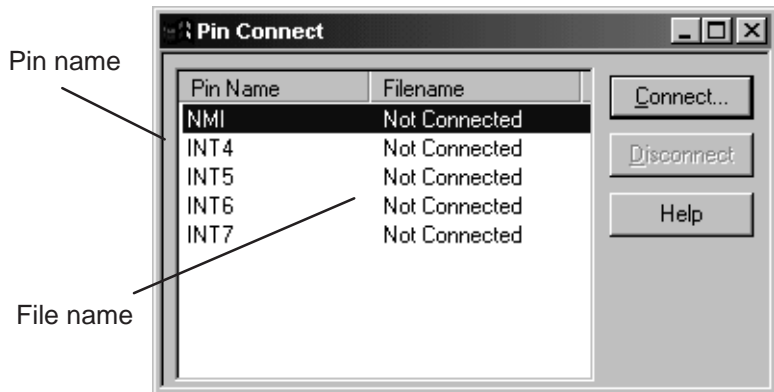
Step 3: Select the Pin name and click Connect.

Step 4: Load your program.

Step 5: Run your program.

For detailed information on the Pin Connect tool, see the Pin Connect topics provided in the online help: Help→Contents→Pin Connect.

4.13 Port Connect

You can use the Port Connect tool to access a file through a memory address. Then, by connecting to the memory (port) address, you can read data in from a file, and/or write data out to a file.

To connect a memory (port) address to a data file, follow these steps:

Step 1: From the Tools menu, select Port Connect.

This action displays the Port Connect window and starts the Port Connect tool.
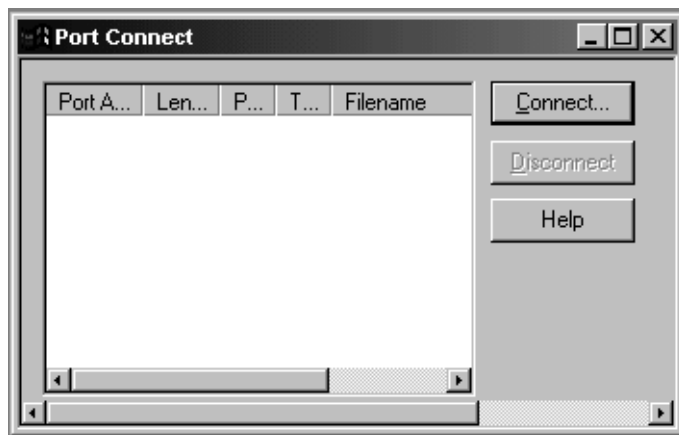
Step 2: Click the Connect button.

This action opens the Connect dialog box.
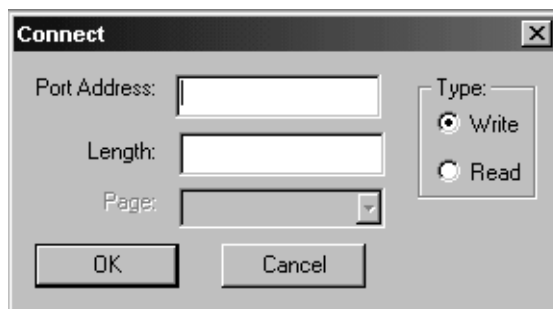
Step 3: In the Port Address field, enter the memory address.

This parameter can be an absolute address, any C expression, the name of a C function, or an assembly language label. If you want to specify a hex address, be sure to prefix the address number with 0x; otherwise, CCS IDE treats the number as a decimal address.

Step 4: In the Length field, enter the length of the memory range.

The length can be any C expression.

Step 5: In the Page field (C5000 only), choose type of memory (program or data) that the address occupies:

To identify this page. . .	use this value.
Program memory	Prog
Data memory	Data
I/O space	I/O

Step 6: In the Type field, select the Write or Read radio button, depending on whether you want to read data from a file or write data to a file.

Step 7: Click OK.

This action displays the Open Port File window.

Step 8: Select the data file to which you want to connect and click Open.

The file is accessed during an assembly language read or write of the associated memory address. Any memory address can be connected to a file. A maximum of one input and one output file can be connected to a single memory address; multiple addresses can be connected to a single file.

For detailed information on the Port Connect tool, see the Port Connect topics provided in the online help: Help→Contents→Port Connect.

4.14 Data Converter

The Data Converter Plug–In (DCP) from Texas Instruments allows fast and easy software development for data converters attached to a Digital Signal Processor (DSP).

Open the Data Converter Support Window

Select Tools→Data Converter Support.

Figure 4–6. Data Converter Support Window
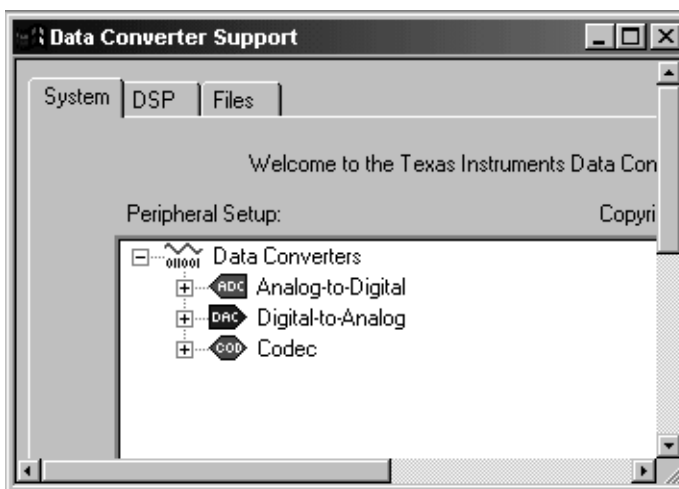
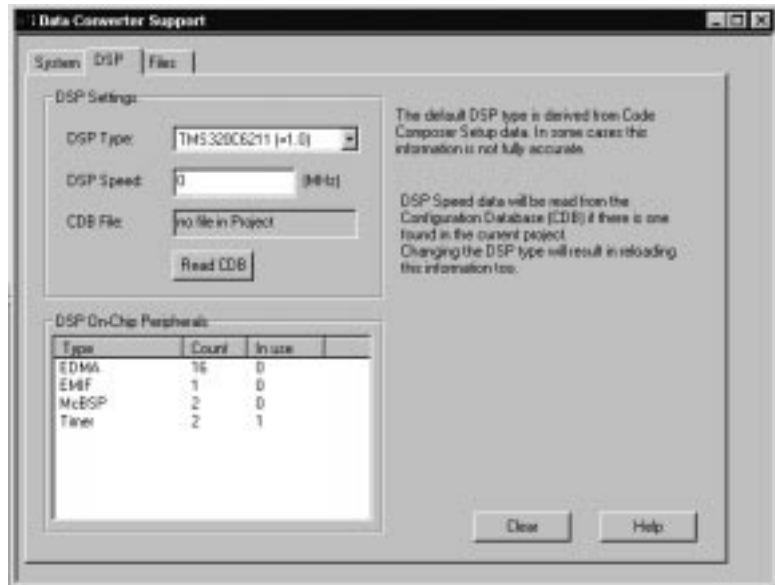For additional information, see the Data Converter portion of the CCS online help.

Configure Your System to Use the Data Converter Plug-in

To configure your system to use the Data Converter Plug-in:

Step 1: System tab. Add devices that are connected to the DSP.

Step 2: DSP tab. Specify required DSP configuration data.

Step 3: "Device Name." Configure your ADC(s), DAC(s) and/or Codec(s).

Step 4: **CCS IDE.** Open a project in CCS IDE.

Step 5: **Files tab.** Generate the configuration files.

Several source code files – usually C language – are added to your DSP project. These files contain the lowest level of interface software required for the data converter(s). In addition, a set of static configuration data is created in header files to be used by the interface software. The data may also be used by your application. The #define strings should be used in your software in order to allow for configuration independent programming.

Step 6: **CCS IDE.** Use data structures and interface functions as needed.

Step 7: **CCS IDE.** Verify correct operation using CCS debug features.

Step 8: **CCS IDE.** Save a workspace before exiting CCS IDE.

Note:

Interface functions are hardware dependent. They only work if standard hardware interfacing is used. It is always your responsibility to verify the correct operation of hardware and software in your system.

Code Composer Studio Optimization Tools

This chapter applies to all platforms using Code Composer Studio™ *(CCS) IDE. However, not all devices have access to all the tools discussed in this chapter. For a complete listing of the tools available to you, see the online help and online documentation provided with CCS IDE.*

Profiling helps you determine how long a processor spends in each section of a program. Using profiling can help you make your programs more efficient. This chapter discusses the CCS profiling tools and shows you how to use them.

5.1 Profiler

Profiling helps reduce the time it takes to identify and eliminate performance bottlenecks. The profiler analyzes program execution and shows you where your program is spending its time. For example, a profile analysis can report how many cycles a particular function takes to execute and how often it is called.

Profiling helps you to direct valuable development time toward optimizing the sections of code that most dramatically affect program performance.

Start a new session each time you want to perform profile analysis on an executable program. When you want to profile more than one executable program, multiple profile sessions can be launched.

To start a new profile session:

Step 1: Select Profiler→Start New Session.
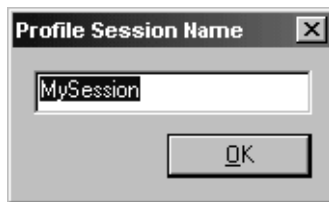
Step 2: In the Profile Session Name dialog box, enter a name for this profile session. By default, the name MySession appears.

Step 3:   Click OK.

A Profile View window appears within the CCS window.

Drag-and-drop any function from your code into the Profile View window.
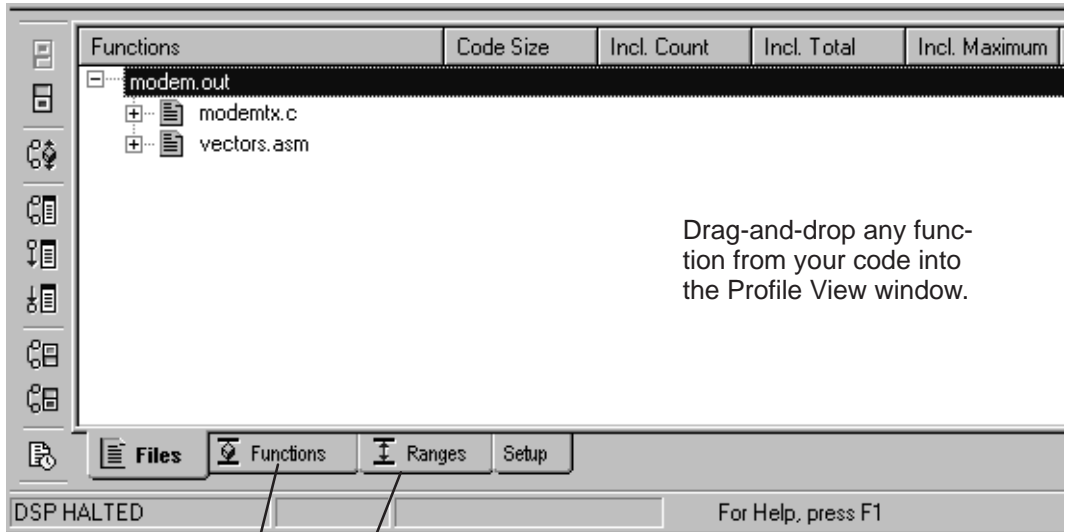
Profile by Function

Profile by Range

Note:

The new profiler allows you to exclude code ranges from being profiled. In other words, you can use Start and End points to exclude certain parts of your code found within the profiling range from being counted as part of the profiling data.

Step 4:   Select File→Load Program.

Step 5:   In the Load Program dialog box, specify the output file to be profiled and click Open.

As a prerequisite to performing profile analysis, the program must be built with symbolic debugging information.

For detailed information on the Profiler tool, see the Profiler topics provided in the online help: Help→Contents→Profiling Code Execution.

5.2   Profile Based Compiler (PBC)

This tool is only available to TMS320C6000   devices.

PBC is a tool that enables you to more quickly and easily optimize the trade-off between code size and cycle count for your application. Using a variety of profiling configurations, PBC will profile your application, collect data on individual functions, and determine the best combinations of compiler options. PBC will then produce a graph of these configurations, allowing you to choose the configuration that best fits your needs.

To understand the behavior of your application and the effects that different compiler options have, PBC must compile and run your application under a number of different profiling configurations.

By default, PBC has 5 different Profiling Configurations defined, ranging from optimize for maximum speed to optimize for minimum code size. By default, 3 Profiling Configurations are activated. If a Profiling Configuration is activated, PBC will use it when profiling your application. A deactivated Profiling Configuration is available, but will not but used until activated. You can also create your own Profiling Configurations. This allows you to define and maintain Profiling Configurations in addition to the defaults that are defined by PBC.

When using PBC with your application, you may want to increase the number of activated Profiling Configurations, and even define some of your own. The more Profiling Configurations used, the longer PBC takes to build and profile your application. This gives PBC more options in finding an optimal build of your application, relevant to your speed or code size constraints.

Enabling Profile Configurations

Step 1: Select PBC→Enable.

Step 2: Select PBC→Profiling Configurations.

The Profiling configurations with check boxes are activated, and should look similar to this:
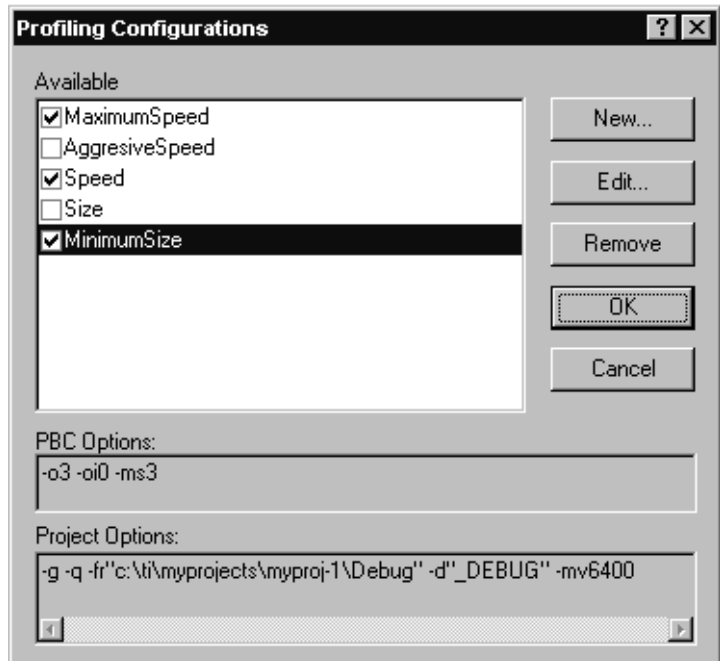
Step 3: Make changes to the profiling configurations if needed.

Step 4: Click OK to close the dialog box.

You will be asked if you want to rebuild your application with the new Configurations, choose Yes if you made any changes. Choose No if you did not make changes. If you choose cancel, you will go back to the profiling configurations dialog.

Building Profile Configurations

Now that you have determined which Profiling Configurations will be used, you can build and profile your application.

Step 1: From the PBC menu, chose ReProfile All.

Step 2: This step will take some time (as long as 30 minutes).

In the CCS output window, the PBC tab shows build progress. When PBC is finished profiling your application, a window like the following one is shown:
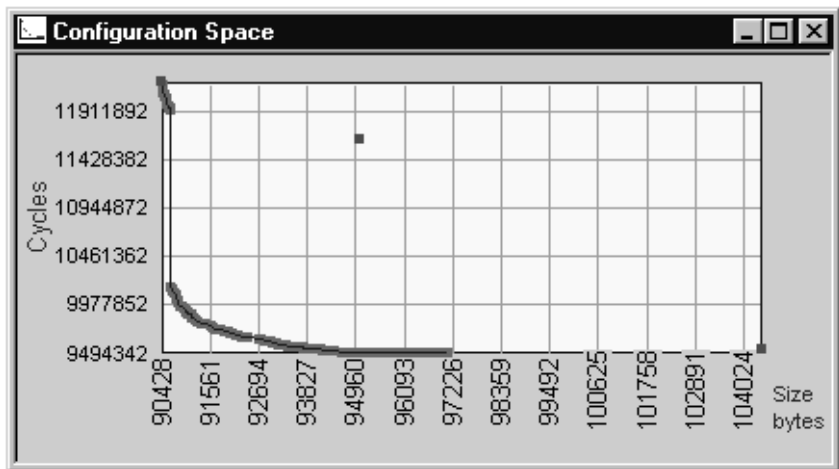
PBC will not build and profile a configuration if it has not changed since the last time it was used. You can force PBC to rebuild and profile Configurations by selecting ReProfile All from the PBC menu.

Once PBC has built and profiled all active profiling configurations, it will have collected profile information on how each function performed under each configuration. PBC uses this information to determine all of the useful configurations that can be used to build your application. A configuration is useful if no other configurations results in a version of your application that is both smaller and faster.

The set of all useful configurations are graphed in purple in the Configuration Space window. You can now select one (or more) of the configurations to use for your application.

For detailed information on the Profiler tool, see the Profiler topics provided in the online help: Help→Contents→Profile Based Compiler.

Figure 5–1. Finished PBC Application
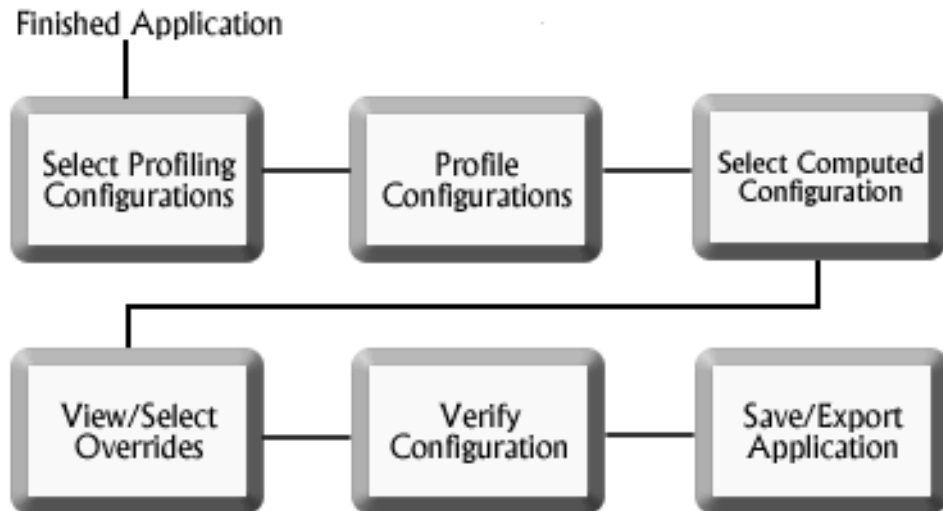
Code Composer Studio
Real-time Kernel and Analysis

This chapter applies to all platforms using Code Composer Studio™
(CCS) IDE. However, not all devices have access to all of the tools discussed
in this chapter. For a complete listing of the tools available to you, see the on-
line help and online documentation provided with CCS IDE.

CCS IDE includes a real-time kernel and real-time analysis features that
simplify development and debugging of complex applications. The features
covered in this chapter are:

❏   DSP/BIOS™ kernel

❏   RTDX™ (Real-Time Data Exchange) technology

To read more about these features, or to determine if you have access to these
features, see the online help and/or tutorial provided with CCS IDE.

6.1 DSP/BIOS Kernel

DSP/BIOS kernel is a scalable real-time kernel. It is designed for applications that require real-time scheduling and synchronization, host-to-target communication, along with real-time instrumentation. DSP/BIOS kernel provides preemptive multi-threading, hardware abstraction, and real-time analysis.

DSP/BIOS kernel is packaged as a set of modules that can be linked into an application; applications include only those functions of the DSP/BIOS kernel that are referenced (directly or indirectly) by the application. In addition, the DSP/BIOS Configuration Tool allows you to optimize code size and speed by disabling DSP/BIOS kernel features not used in their programs.

You can use DSP/BIOS kernel to instrument any application to be probed, traced, and monitored in real-time. Programs that use the DSP/BIOS Configuration Tool to take advantage of the multi–threading capabilities of DSP/BIOS kernel are implicitly instrumented.

DSP/BIOS kernel is integrated with CCS IDE and requires no runtime license fees.
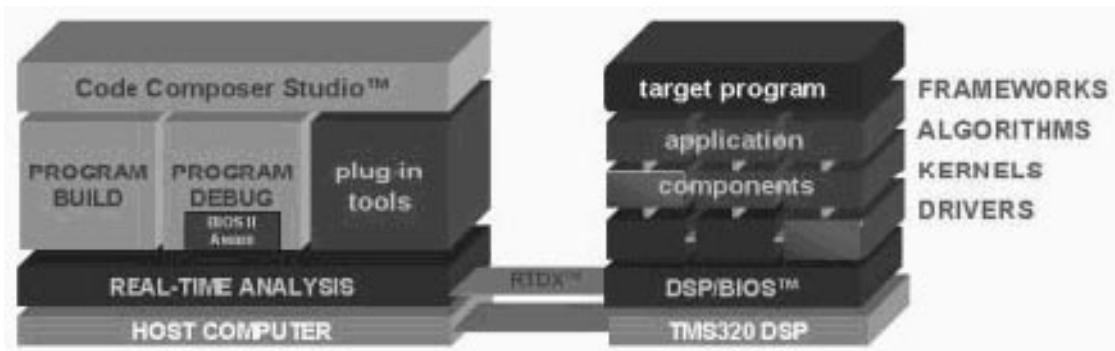
For details on on using DSP/BIOS kernel, see the Help→Contents→DSP/BIOS or Help→Tutorial→Using DSP/BIOS.

DSP/BIOS kernel includes the following components:

❑ **DSP/BIOS Configuration Tool.** This tool allows you to create and configure the DSP/BIOS kernel objects used by your program. You can also use this tool to configure memory, thread priorities, and interrupt handlers.

❑ **DSP/BIOS Real-time Analysis Tools.** These windows allow you to view program activity in real-time. For example, the Execution Graph shows a diagram of thread activity.

❑ **DSP/BIOS Kernel.** The DSP/BIOS kernel implements run-time services that the target application invokes through DSP/BIOS APIs.

❑ **Chip Support Library:** The Chip Support Library provides macros and functions that simplify configuration and management of on-chip peripherals. See Chapter 7 for more information.

DSP/BIOS Configuration Tool

The DSP/BIOS Configuration Tool tightly integrates with CCS IDE. This tool enables developers to select and deselect kernel modules, and control a wide range of configurable parameters accessed by the DSP/BIOS kernel at run–time as shown in the figures below. A file of data tables generated by the tool ultimately becomes an input to the program linker.

Figure 6–1. DSP/BIOS Configuration Window
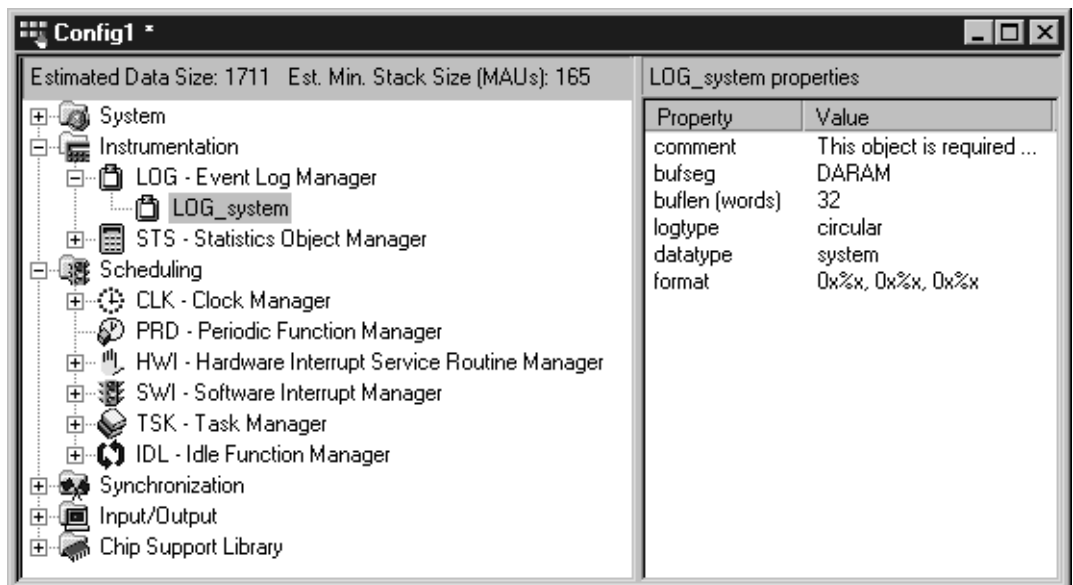
The DSP/BIOS Configuration Tool (see Figure 6–1) serves as a special-purpose visual editor for creating and assigning attributes to individual run-time kernel objects (threads, streams, etc.) used by the target application program in conjunction with DSP/BIOS API calls. The Configuration Tool provides developers the ability to statically declare and configure DSP/BIOS kernel objects during program development rather than during program execution. Declaring these kernel objects through the Configuration Tool produces static objects which exist for the duration of the program. DSP/BIOS kernel also allows dynamic creation and deletion for many of the kernel objects during program execution. However, dynamically created objects require additional code to support the dynamic operations. Statically declared objects minimize memory footprint since they do not include the additional create code.

Another important benefit of static configuration is the potential for static program analysis by the DSP/BIOS Configuration Tool. In addition to minimizing the target memory footprint, the DSP/BIOS Configuration Tool provides the means for early detection of semantic errors through the validation of object attributes, prior to program execution. When the configuration tool is aware of all target program objects prior to execution, it can accurately compute and report such information as the total amount of data memory and stack storage required by the program.

Creating DSP/BIOS Configuration Files

To create DSP/BIOS configuration files:

Step 1: Within Code Composer Studio, choose File→New→DSP/BIOS Configuration.

The New Configuration window displays.

Available DSP/BIOS configurations for your platform

View detailed list
View list as small icons
View list as large icons

Description of the selected configuration
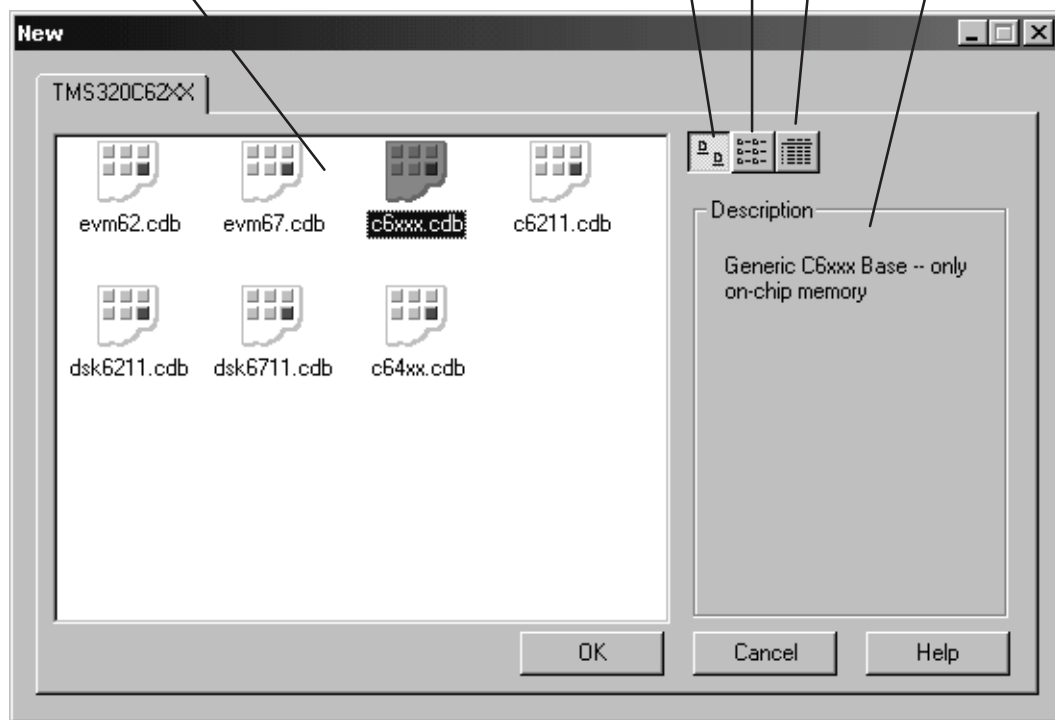
Step 2: Select a Configuration template.

If your board is not listed, you can create and add a custom template to this list.

Step 3: Click OK to create the new configuration.

The Configuration window displays.
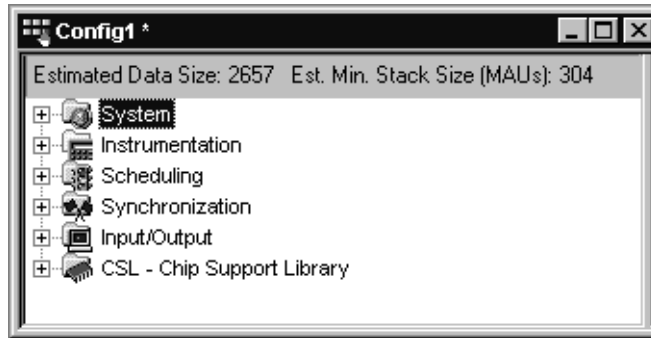
Step 4: In the Configuration window, perform the following tasks as required by your application:

- Create objects to be used by the application.

- Name the objects.

- Set global properties for the application.

- Modify module manager properties.

- Modify object properties.

- Set priorities for software interrupts and tasks.

See Help→Contents→DSP/BIOS→DSP/BIOS API Modules for details on implementation of APIs.

Step 5: Save the configuration.

Step 6: Add the DSP/BIOS configuration file(s) to your project as described in the next procedure.

Adding DSP/BIOS Configuration files to your project

After you save a DSP/BIOS configuration file, follow these steps to add files to your Code Composer Studio project.

Step 1: If it is not already open, use Project→Open to open the project with Code Composer Studio.

Step 2: Choose Project→Add Files to Project. In the Files of type box, select Configuration File (*.cdb). Select the .cdb file you saved and click Open.

Adding the .cdb file to a project automatically adds the following file to the Project View folders:

- program.cdb in the DSP/BIOS Config folder

- programcfg.s62 in the Generated Files folder

- programcfg_c.c in the Generated Files folder

Step 3: Choose Project→Add Files to Project again. In the Files of type box, select Linker Command File (*.cmd). Select the *cfg.cmd file the Configuration Tool generated when you saved the configuration file and click Open.

Step 4: If your project already contained a linker command file, Code Composer Studio warns you that a project can only contain one linker command file and asks if you want to replace the existing file.

Step 5: If your project includes the vectors.asm source file, right-click on the file and choose Remove from project in the shortcut menu. Hardware interrupt vectors are automatically defined in the configuration file.

Step 6: If your project includes the rts*xxxx*.lib file (where *xxxx* is your device or device's generation), right–click on the file and choose Remove from project in the shortcut menu. This file is automatically included by the linker command file created from your configuration.

These steps can be used whenever you want to convert an existing program to one that can call DSP/BIOS API functions.

DSP/BIOS Real-time Analysis Tools

The DSP/BIOS Real-Time Analysis (RTA) features, shown in Figure 6–2, provide developers and integrators unique visibility into their application by allowing them to probe, trace, and monitor a DSP application during its course of execution. These utilities, in fact, piggyback upon the same physical JTAG connection already employed by the debugger, and utilize this connection as a low-speed (albeit real-time) communication link between the target and host.

Figure 6–2. Real-Time Capture and Analysis
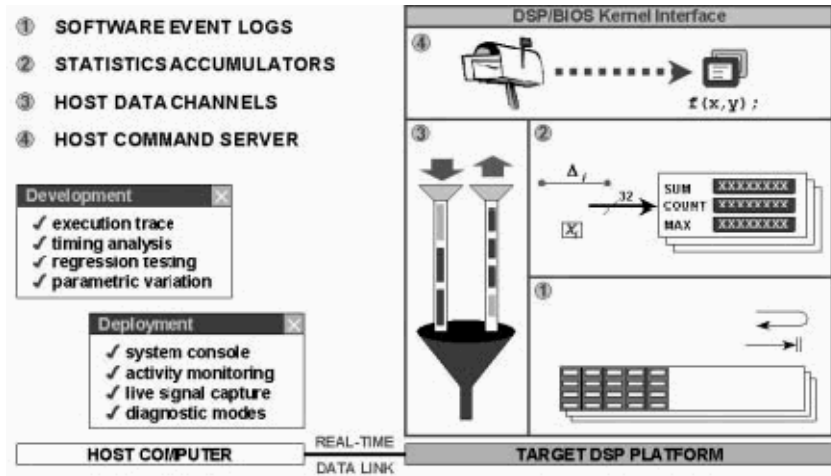
DSP/BIOS RTA requires the presence of the DSP/BIOS kernel within the target system. In addition to providing run-time services to the application, DSP/BIOS kernel provides support for real-time communication with the host through the physical link. By simply structuring an application around the DSP/BIOS APIs and statically created objects that furnish basic multitasking and I/O support, developers automatically instrument the target for capturing and uploading the real-time information that drives the visual analysis tools inside CCS IDE. Supplementary APIs and objects allow explicit information capture under target program control as well. From the perspective of its hosted utilities, DSP/BIOS affords several broad capabilities for real-time program analysis:

The DSP/BIOS Real-Time Analysis tools can be accessed through the DSP/BIOS toolbar.

Figure 6–3. DSP/BIOS Toolbar

Kernel/Object View

CPU Load Graph

Execution Graph

RTA Control Panel

Host Channel Control

Statistics View

Message Log
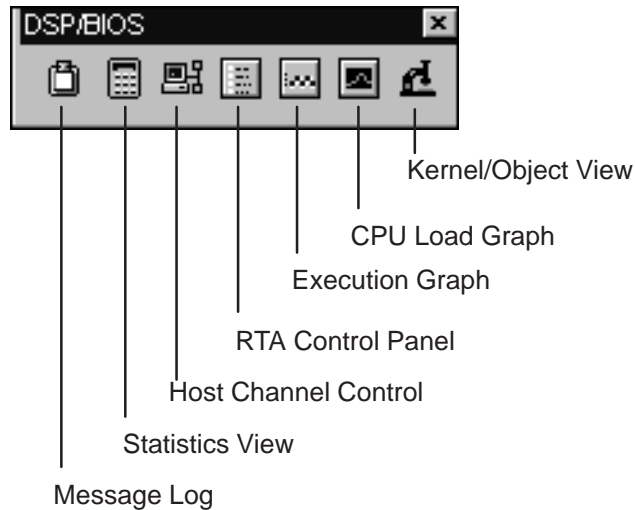
❏ **Message Event Logs.** Capable of displaying time-ordered sequences of events written to kernel log objects by independent real-time threads, tracing the program's overall flow of control. The target program logs events explicitly through DSP/BIOS API calls or implicitly by the underlying kernel when threads become ready, dispatched, and terminated.

❏ **Statistics Accumulators.** Capable of displaying summary statistics amassed in kernel accumulator objects, reflecting dynamic program elements ranging from simple counters and time-varying data values, to elapsed processing intervals of independent threads. The target program accumulates statistics explicitly through DSP/BIOS API calls or implicitly by the kernel when scheduling threads for execution or performing I/O operations.

❏ **Host Data Channels.** Capable of binding kernel I/O objects to host files providing the target program with standard data streams for deterministic testing of algorithms. Other real-time target data streams managed with kernel I/O objects can be tapped and captured on-the-fly to host files for subsequent analysis.

❏ **Host Command Server (RTA Control Panel).** Capable of controlling the real-time trace and statistics accumulation in target programs. In effect, this allows developers to control the degree of visibility into the real-time program execution.

Note:

When used in tandem with the CCS IDE standard debugger during software development, the DSP/BIOS real-time analysis tools provide critical visibility into target program behavior at exactly those intervals where the debugger offers little or no insight – during program execution. Even after the debugger halts the program and assumes control of the target, information already captured through DSP/BIOS can provide invaluable insights into the sequence of events that led up to the current point of execution.

Later in the software development cycle, regular debuggers become ineffective for attacking more subtle problems arising from time-dependent interaction of program components. The DSP/BIOS real-time analysis tools subsume an expanded role as the software counterpart of the hardware logic analyzer.

This dimension of DSP/BIOS becomes even more pronounced after software development concludes. The embedded DSP/BIOS kernel and its companion host analysis tools combine to form the necessary foundation for a new generation of manufacturing test and field diagnostic tools. These tools will be capable of interacting with application programs in operative production systems through the existing JTAG infrastructure.

The overhead cost of using DSP/BIOS is minimal, therefore instrumentation can be left in to enable field diagnostics, so that developers can capture and analyze the actual data that caused the failures.

DSP/BIOS Kernel

The DSP/BIOS kernel implements run-time services that the target application program invokes through DSP/BIOS APIs.

Individual DSP/BIOS modules in general will manage one or more instances of a related class of objects, sometimes referred to as kernel objects, and will rely upon global parameter values to control their overall behavior.

Developers can statically declare and configure many of these objects using the DSP/BIOS Configuration Tool. Developers may also declare and configure many of these objects dynamically within their programs.

C, C++, and assembly language programs can call over 150 DSP/BIOS API functions. DSP/BIOS is packaged as a set of modules that can be linked into an application; applications include only those functions of the DSP/BIOS that are referenced (directly or indirectly) by the application. In addition, the DSP/BIOS Configuration Tool allows you to optimize code size and speed by disabling DSP/BIOS features not used in their programs.

Execution Threads

When applications are organized as independent paths of execution, developers can place structure and order into them (see Figure 6–4). DSP/BIOS execution threads are independent paths of execution that execute an independent stream of DSP instructions. An execution thread is a single point of control that may contain an ISR, subroutine, or a function call. For example, a hardware interrupt is a thread, and it performs the ISR when triggered.

Figure 6–4. DSP/BIOS Execution Threads
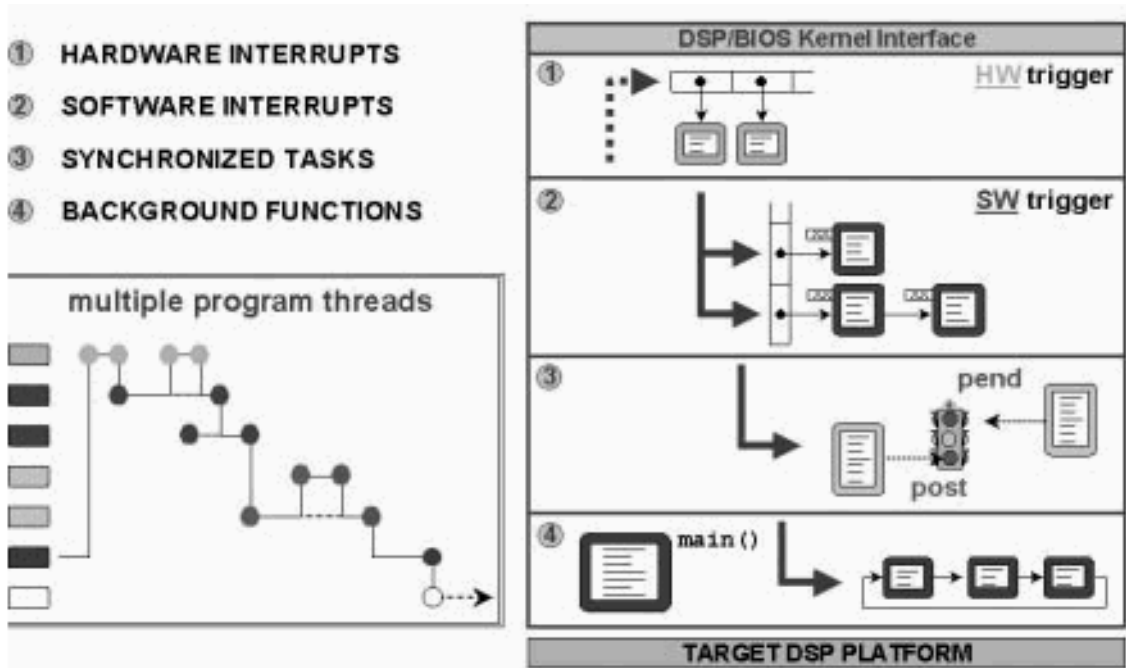
Multithreaded applications can run on single processor systems by allowing higher-priority threads to preempt lower–priority threads. DSP/BIOS provides 30 levels of priority, divided over four distinct classes of execution threads (see Figure 6–4). DSP/BIOS also provides services to support the synchronization of, and communication between, execution threads. Multirate processing maps well onto multithreaded systems.

With the exception of the background idle processing thread, each thread type supports multiple levels of priority. DSP/BIOS provides choices; it allows developers to use the optimum thread-types for their application and not bend their application to fit a certain model. DSP/BIOS developers have the flexibility to mix and match the objects in the run-time library that are best suited for the application. DSP/BIOS is completely scalable and only those modules that have been selected link with the application, minimizing resource requirements.

For more information on using DSP/BIOS, see the Help→Contents→DSP/BIOS or Help→Tutorial→Using DSP/BIOS.

6.2 RTDX Technology

DSP/BIOS Real-Time Analysis (RTA) facilities utilize the Real-Time Data Exchange (RTDX) link to obtain and monitor target data in real-time. You can utilize the RTDX link to create your own customized interfaces to the DSP target by using CCS IDE's RTDX API Library.

Real-time data exchange (RTDX) allows system developers to transfer data between a host computer and target devices without interfering with the target application. This bi-directional communication path provides for data collection by the host as well as host interaction with the running target application. The data collected from the target may be analyzed and visualized on the host. Application parameters may be adjusted using host tools, without stopping the application. RTDX also enables host systems to provide data stimulation to the target application and algorithms.

RTDX consists of both target and host components. A small RTDX software library runs on the target application. The target application makes function calls to this library's API in order to pass data to or from it. This library makes use of a scan-based emulator to move data to or from the host platform via a JTAG interface. Data transfer to the host occurs in real-time while the target application is running.

On the host platform, an RTDX Host Library operates in conjunction with CCS IDE. Data visualization and analysis tools communicate with RTDX through COM APIs to obtain the target data and/or to send data to the DSP application.

The host library supports two modes of receiving data from a target application: Continuous and Non Continuous. In Continuous mode, the data is simply buffered by the RTDX Host Library and is not written to a log file. Continuous mode should be used when the developer wants to continuously obtain and display the data from a target application, and does not need to store the data in a log file. In Non Continuous mode, data is written to a log file on the host. This mode should be used when developers want to capture a finite amount of data and record it in a log file.

For details on using RTDX, see the Help→Contents→RTDX or Help→Tutorial→RTDX Tutorial.

RTDX Data Flow

RTDX forms a two-way data pipe between a target application and a host cli-ent. This data pipe consists of a combination of hardware and software compo-nents as shown below.

Figure 6–5. RTDX Data Flow
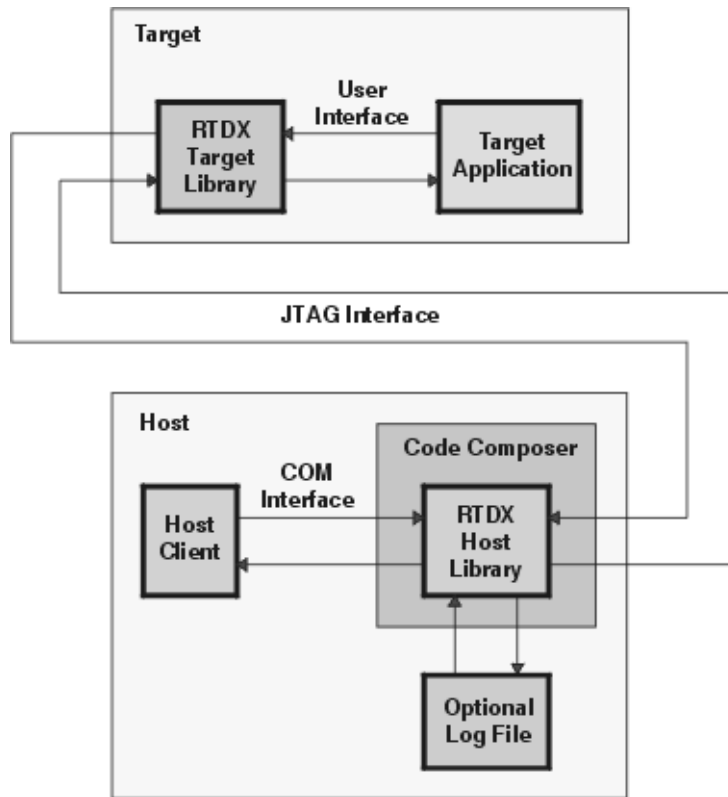
Configuring RTDX Graphically

The RTDX tools within CCS IDE allow you to configure RTDX graphically, set up RTDX channels, and run diagnostics on RTDX. These tools allow you to enhance RTDX functionality when transmitting data.

RTDX has three menu options:

❑ Diagnostics Control
❑ Configuration Control
❑ Channel Viewer Control

Figure 6–6. RTDX Menu
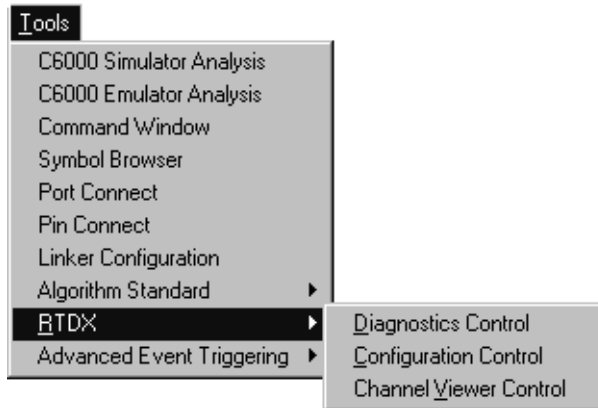
Diagnostics Control

RTDX provides the RTDX Diagnostics Control to verify that RTDX is working correctly on your system. The diagnostic tests test the basic functionality of target–to–host transmission and host–to–target transmission.

To open the RTDX Diagnostics Control in CCS IDE, select Tools→RTDX→Diagnostics Control. The Diagnostics Control window appears near the bottom of the CCS IDE interface.

Figure 6–7. RTDX Diagnostics Window
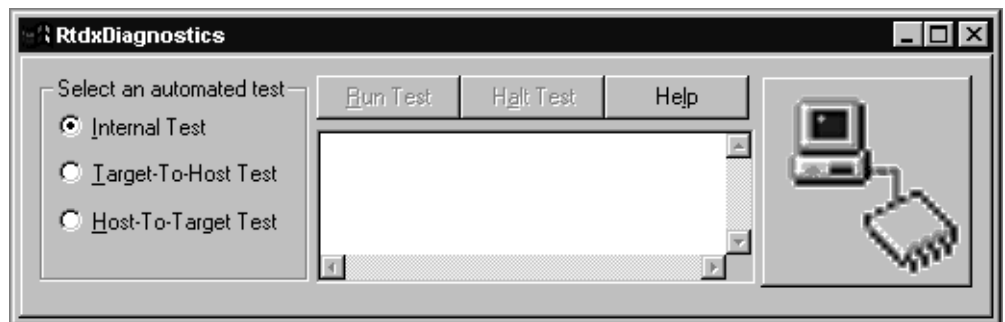
Configuration Control

RTDX provides the RTDX Configuration Control to configure and control RTDX graphically. Configuration Control is the main RTDX window. It allows you to do the following:

❑ View the current RTDX configuration settings

❑ Enable or disable RTDX

❑ Access the RTDX Configuration Control Properties page to reconfigure RTDX and select port configuration settings

To open the RTDX Configuration Control in CCS IDE, select Tools→RTDX→Configuration Control. The Configuration Control window displays.

Figure 6–8. RTDX Config Window

Click this button to reconfigure properties
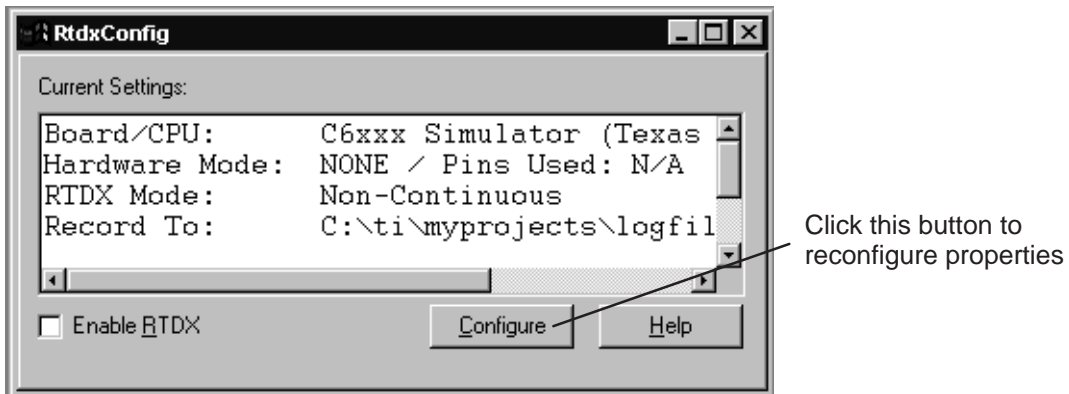
Channel Viewer Control

RTDX provides the RTDX Channel Viewer Control to set up RTDX channels. The RTDX Channel Viewer Control allows you to:

❏ Add or remove target–declared channels to the viewable list.

❏ Enable or disable channels once they have been added to the list.

To open the RTDX Channel Viewer Control in CCS IDE, select Tools→RTDX→Channel Viewer Control. The Channel Viewer Control window displays.

Figure 6–9. RTDX Channel Viewer Window
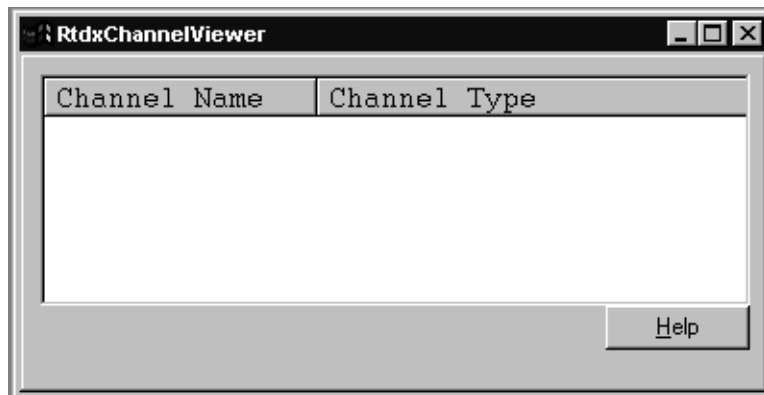
Transmit a Single Integer to the Host

The basic function of RTDX is to transmit a single integer to the host. The following steps provide an overview of the process of sending data from the target to the host and from the host to the target. For specific commands and details on transmitting different types of data, see the Help→Contents→RTDX or Help→Tutorial→RTDX Tutorial.

To send data from your target application to the host:

Step 1: Prepare your target application to capture real-time data.

This involves inserting specific RTDX syntax into your application code to allow real-time data transfer from the target to the host. Although the process for preparing a target application is the same for all data types, different data types require different function calls for data transfer. Therefore, sending an integer to the host requires you to add a function call that is specific to only transmitting a single integer as compared to sending an array of integers to the host.

Step 2: Prepare your host client to process the data.

This involves instantiating one RTDX object for each desired channel, opening a channel for the objects specified, and calling any other desired functions.

Step 3: Start CCS IDE.

Step 4: Load your target application onto the TI processor.

Step 5: Enable RTDX in CCS IDE: Tools→RTDX→Configuration Control.

The Configuration Control window displays.

Step 6: Run your target application to capture real-time data and send it to the RTDX Host Library.

Step 7: Run your host client to process the data.

For details on using RTDX, see the Help→Contents→RTDX or Help→Tutorial→RTDX Tutorial.

Transmit Data from the Host to the Target

A client application can send data to the target application by writing data to the target. Data sent from the client application to the target is first buffered in the RTDX Host Library. The data remains in the RTDX Host Library until a request for data arrives from the target. Once the RTDX Host Library has enough data to satisfy the request, it writes the data to the target without interfering with the target application.

The state of the buffer is returned into the variable buffer state. A positive value indicates the number of bytes the RTDX Host Library has buffered, which the target has not yet requested. A negative value indicates the number of bytes that the target has requested, which the RTDX Host Library has not yet satisfied.

To send data from a host client to your target application:

Step 1: Prepare your target application to receive data.

This involves writing a simple RTDX target application that reads data from the host client.

Step 2: Prepare your host client to send data.

This involves instantiating one RTDX object for each desired channel, opening a channel for the objects specified, and calling any other desired functions.

Step 3: Start CCS IDE.

Step 4: Load your target application onto the TI processor.

Step 5: Enable RTDX in CCS IDE: Tools→RTDX→Configuration Control.

The Configuration Control window displays.

Step 6: Run your target application.

Step 7: Run your host client.

For details on using RTDX, see the Help→Contents→RTDX or Help→Tutorial→RTDX Tutorial.

Code Composer Studio
Chip Support Library Overview

This chapter applies to all platforms using Code Composer Studio ™ *(CCS) IDE. However, not all devices have access to all of the tools discussed in this chapter. For a complete listing of the tools available to you, see the online help and online documentation provided with CCS IDE.*

This chapter introduces the Chip Support Library, briefly describes its architecture, and provides a generic overview of the collection of functions, macros, and constants that help you program DSP peripherals.

For more information on CSL, see the online help provided with CCS IDE.

7.1 Introduction to CSL

The Chip Support Library(CSL) is a fully scalable component of DSP/BIOS that provides C-program functions to configure and control on-chip peripherals. It is intended to simplify the process of running algorithms in a real system. The goal is peripheral ease of use, shortened development time, portability, hardware abstraction, and a small level of standardization and compatibility among devices.

How the CSL Benefits You

The CSL benefits you in the following ways:

❑ **Standard Protocol to Program Peripherals**

CSL provides a higher-level programming interface for each on-chip peripheral. This includes data types and macros to define peripheral register configuration, and functions to implement the various operations of each peripheral.

❑ **Basic Resource Management**

Basic resource management is provided through the use of open and close functions for many of the peripherals. This is especially helpful for peripherals that support multiple channels.

❑ **Symbol Peripheral Descriptions**

As a side benefit to the creation of CSL, a complete symbolic description of all peripheral registers and register fields has been created. It is suggested that you use the higher level protocols described in the first two bullets, as these are less device specific, making it easier to migrate your code to newer versions of DSP's.

The DSP/BIOS configuration tool provides a graphical user interface for device configuration via the CSL. The CSL tree of the configuration tool allows the pre-initialization of some peripherals by generating C files using CSL APIs. The peripherals are pre-configured with the pre-defined configuration objects.

Note:

If you utilize CSL without use of the DSP/BIOS Configuration tool, CSL can conflict with DSP/BIOS functionality.

7.2 Introduction to the DSP/BIOS Configuration Tool: CSL Tree

The DSP/BIOS Configuration Tool allows you to access the CSL graphical interface and configure some of the on-chip peripherals. Each peripheral is represented as a subdirectory of the CSL Tree as shown in Step 3.

The work-flow consists of the following steps:

Step 1: Creation of the DSP/BIOS configuration file (.cdb).
File→New→DSP/BIOS Configuration.

The New configuration window displays.
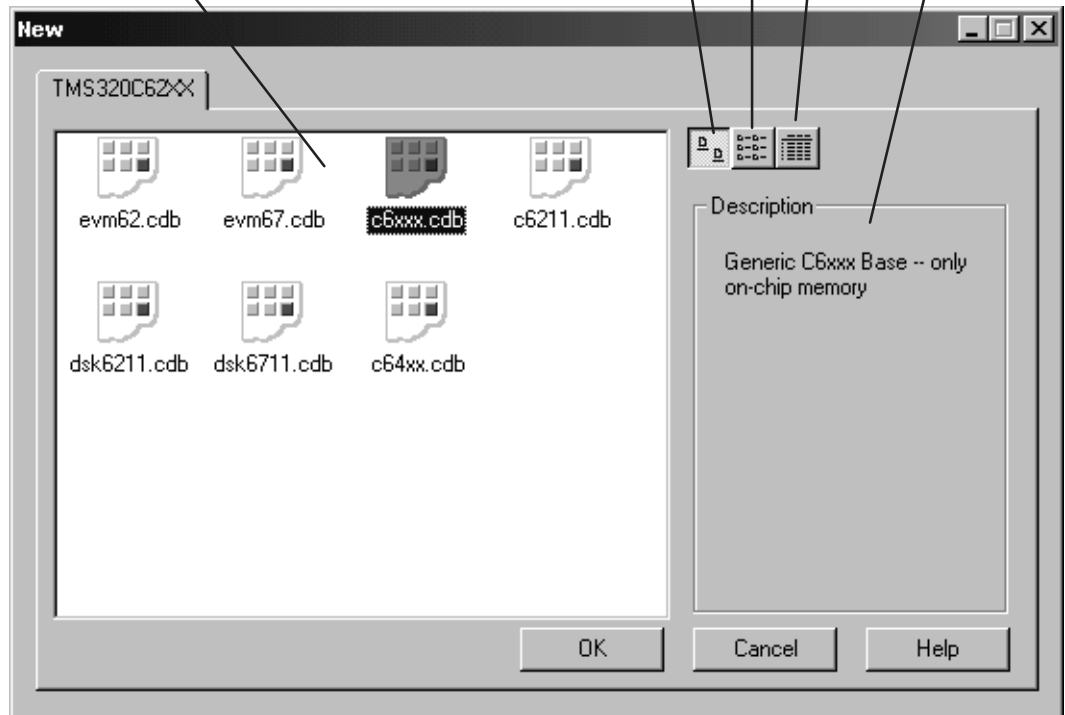
Step 2: Select the appropriate configuration for your target and click OK.

The Config window displays.
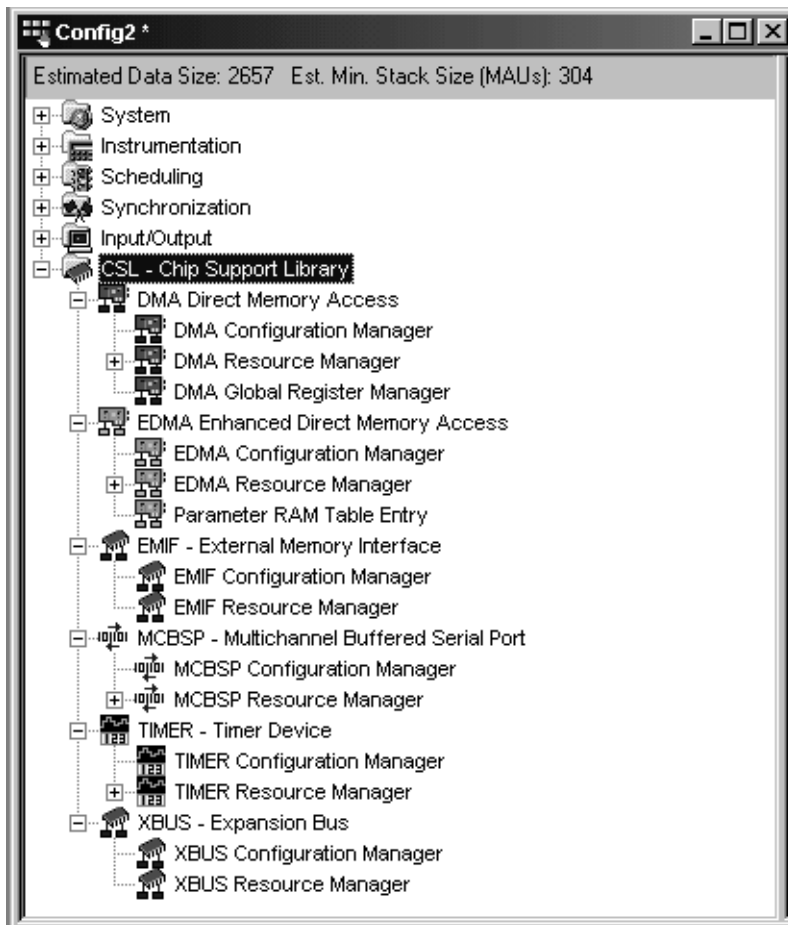
Step 3: Configure the on-chip peripherals through the CSL hierarchy tree.

Add configuration icons and then set the values and parameters for each configuration.
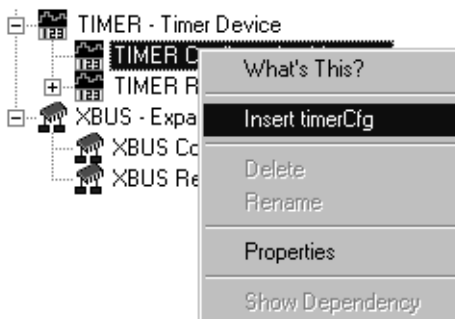
You can also set global properties. Under the System icon in the Configuration window, highlight Global Settings. Right-click and select Properties from the pop-up menu. The Global Settings Properties window displays.
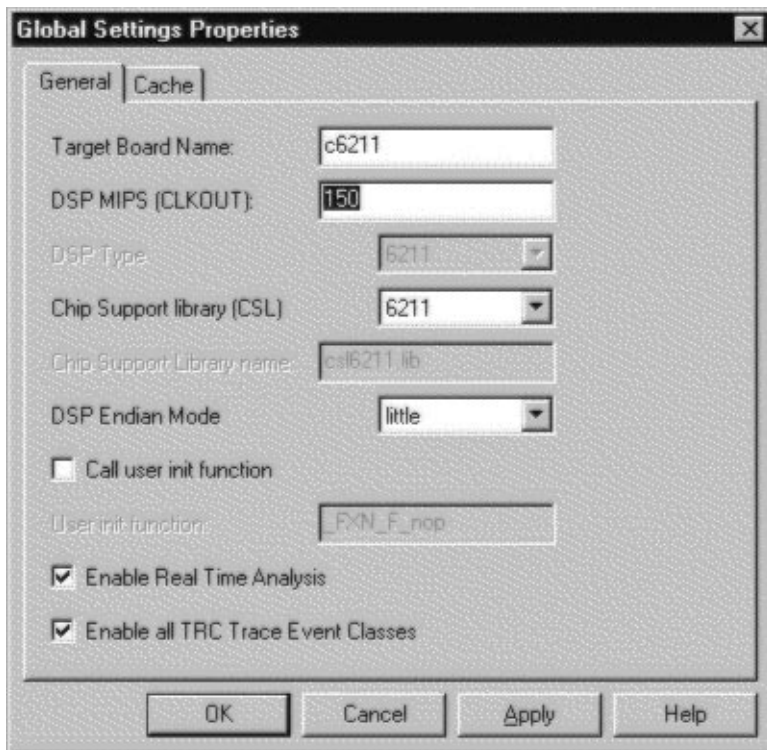
Step 4: C-code files are automatically generated when you save the configuration file. These files display in your project window.

Two C files are generated – the header file and the source file.

- Header file: project**cfg.h**
- Source file: project**cfg_c.c**

In these examples, project is the user's cdb file name. The bold characters are attached automatically.

Header file

The header file contains several elements.

❏ The definition of the chip

```
#define CHIP_6201 1
```

❏ The csl header files of the CSL tree

```
#include <csl.h>

#include <csl_dma.h>
#include <csl_emif.h>
#include <csl_mcbsp.h>
#include <csl_timer.h>
0
```

❏ The declaration list of the *variables* handle and configuration names defined in the *project*.cdb. They are declared external and can be used by the user

```
extern far TIMER_Config timerCfg1;
extern far MCBSP_Config mcbspCfg0;

extern TIMER_Handle hTimer1;
extern MCBSP_Handle hMcbsp0;
```

In order to access the predefined handle and configuration objects, the header file has to be included in the user's project C file.

```
/* User's main .c file */
```

The following line is mandatory and must be included in the user⬚s C file:

```
#include <projectcfg.h>
```

Source file

The source file consists of the Include section, the Declaration section, and the Body section.

❑ **Include section**

The source file has access to the data declared in the header file.

```
#include <projectcfg.h>
```

Note: If this line is added before the other csl header files (csl_emif, csl_timer, ...), you are not required to specify the device number under the Project option (–dCHIP_6xxxx not required).

❑ **Declaration section:**

This section describes the configuration structures and the handle objects previously defined in the configuration tool.

The values of the registers reflect the options selected through the Properties pages of each device.

❑ **Body section:**

The body is composed of a unique function, CSL_cfgInit(), which will be called from the user's C file.

The function CSL_cfgInit() allows you to allocate and configure a device by calling the Peripheral_open() and Peripheral_config() APIs.

These two functions are generated when the Open and Enable Pre-initialization options are checked in the Properties page of the related Resource Manager. An exception to this is when the EMIF handle does not exist.

Note: A device can be allocated without being configured.

TMS320 DSP Algorithm Standard

This chapter applies to all platforms using Code Composer Studio™ (CCS) IDE. However, not all devices have access to all of the tools discussed in this chapter. For a complete listing of the tools available to you, see the on-line help and online documentation provided with CCS IDE.

The TMS320 DSP Algorithm Standard is a key ingredient of eXpressDSP. Its coding conventions for algorithm writers ultimately eliminate much of the time–consuming reengineering work required to integrate algorithms into a variety of applications. It achieves this by defining common programming rules, guidelines, and interfaces.

The Algorithm Standard enforces known behaviors, requires documentation of features relevant to integration, and defines interfaces for algorithms to use to request resources. This facilitates the integration and deployment of algorithms in a variety of systems.

This chapter discusses the TMS320 DSP Algorithm Standard and provides resources for algorithm writers to help them create algorithm interfaces.

For more information on the TMS320 DSP Algorithm Standard, see Help→Contents→TMS320 DSP Algorithm Standard.

8.1   TMS320 DSP Algorithm Standard

Texas Instruments tests algorithms for compliance with the TMS320 DSP Algorithm Standard. Algorithms that pass may use the eXpressDSP-compliant logo. By using such algorithms, system integrators can avoid bugs that result from unfounded assumptions by the algorithm about resource availability and calling context.

Rules and guidelines for writing portable code are provided for the following DSP platforms:

❑   TMS320C62x™, C64x™, C67x™

❑   TMS320C54x™, C55x™

❑   TMS320C24x™

The TMS320 DSP Algorithm Standard Developer's Kit included in your CCS installation contains tools to assist both algorithm producers and consumers.

The Developer's Kit includes documents, examples, and supplementary APIs. Example versions are provided to run out of the box on the following platforms:

❑   C54x simulator

❑   C62x simulator

❑   EVM6201

❑   DSK6211

❑   DSK5402

8.2 Resources for Algorithm Writers

By following these steps, you generate the files needed to create eXpressDSP-compliant algorithm interfaces. If you need help while using these steps in CCS IDE, press F1.

Step 1: Select Tools → Algorithm Standard → Template Code Generator.

The Algorithm Standard Code Generator Window displays.
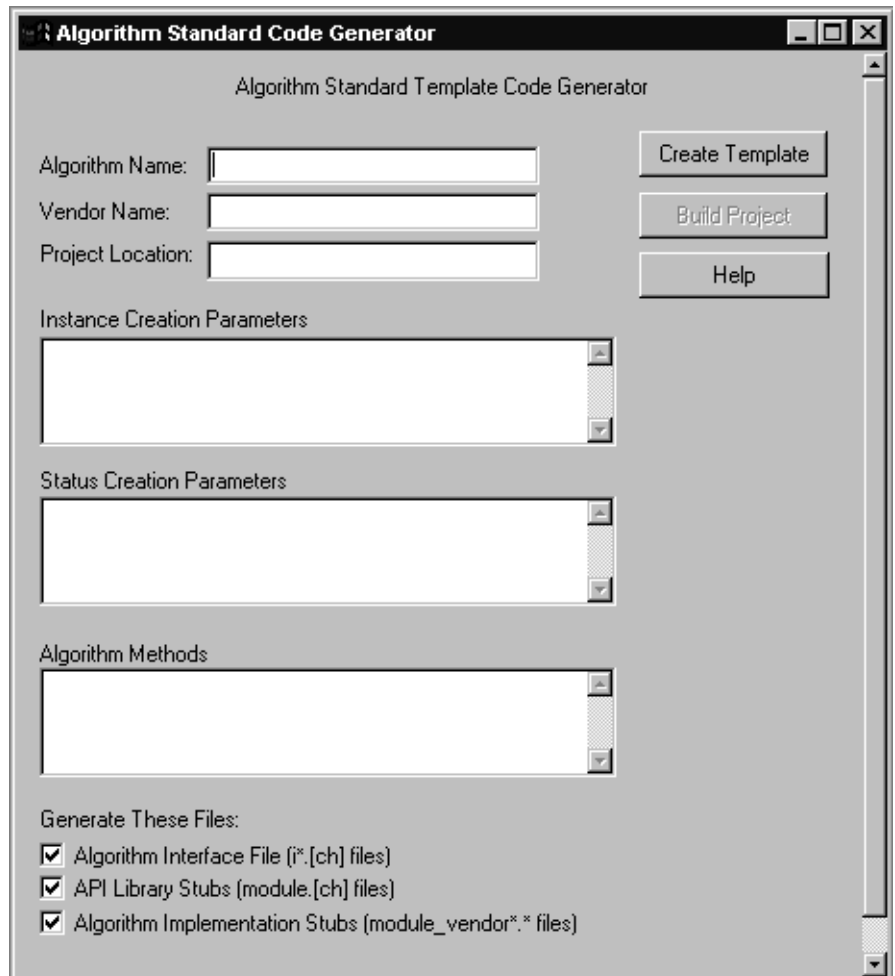
Step 2: Enter values in the fields.

The Algorithm Name, Vendor Name, and Algorithm Methods fields are required.

If all required fields are filled, the Build Project button activates.

TIP: When entering this information, do not use semicolons because the tool adds them automatically.

Step 3: Click Create Template to create your new DSP Algorithm template.

Step 4: Click Build Project.

The Build Progress dialog box and the Output window display.
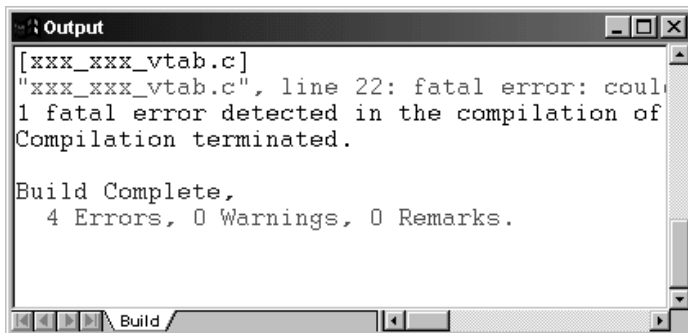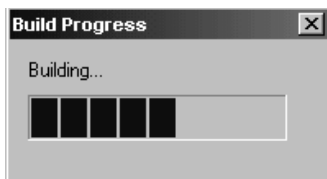
Step 5: The Algorithm Standard files are generated and placed in the project specified by the Project Location field.

Step 6: Add the following files to the project:

- alg-create.c and alg_malloc.c

- The object files containing the original vendor algorithm.

- rts[dsp].lib (the DSP-specific run time system library)

- The framework or application containing the main() function.

Step 7: Change the build options (Project→Options) so the 'Include Search Path' contains the following paths – order is important:

- ■ <path for project directory>

- ■ <path for any header files required for original algorithm>

- ■ c:\ti\xdas\include

Step 8: Edit the i<module>.c file.

This file contains the Params structure default values. Fill in each default value with a reasonable value as the tool puts in zero by default for each parameter.

Step 9: Rename the <MODULE>_<VENDOR>.c file to <MODULE>_<VENDOR>ialg.c and edit the file.

Step 10: Rename the <MODULE>_<VENDOR>_vtab.c file to <MODULE>_<VENDOR>ialgvt.c.

Step 11: Add the following function definitions to the bottom of the <MODULE>_<VENDOR>.h file, before the final #endif statement.

```
/*
 *  ======== <MODULE>_<VENDOR>_init ========
 *  Initialize the <MODULE>_<VENDOR> module as a whole.
 */
Void <MODULE>_<VENDOR>_init(Void);

/*
 *  ======== <MODULE>_<VENDOR>_exit ========
 *  Exit the <MODULE>_<VENDOR> module as a whole.
 */
Void <MODULE>_<VENDOR>_exit(Void);
```

For information about submitting a product for compliance testing, on the TI web site go to the DSP Developers' Village and follow links in the eXpressDSP Compliance Program box.

Index

G

GEL, general extension language   4-24
graphical interface, CSL   7-3
graphs, displaying   4-21

H

hardware breakpoints   4-5
header file, CSL   7-6
hex conversion utility, described   3-7

I

Import Configuration dialog box   1-3, 1-4
import makefile   2-12
interlist utility, described   3-13

K

kernel, DSP/BIOS   6-3, 6-11
keyboard shortcuts, code window   2-14
keywords, code window   2-13

L

library-build utility, described   3-13
linker
    described   3-7
    overview   3-9
    Visual Linker   3-10

M

makefile
    external   2-12
    import   2-12
mnemonic–to–algebraic translator utility   3-7
multiple CCS versions, Component Manager   1-10
multiple tool versions, Component Manager   1-10

N

new project
    create a   2-2
    Project Creation wizard   2-2

O

optimization tools
    Profile Based Compiler   5-4
    Profiler   5-2
optimizer, described   3-13
overview
    assembler   3-8
    debug tools   4-2
    linker   3-9

P

PBC
    building configuration   5-6
    finished application outline   5-7
    Profile Based Compiler   5-4
    profile configuration   5-5
Pin Connect   4-26
Port Connect   4-27
probe points   4-10
    vs. breakpoints   4-10
Profile Based Compiler
    building configuration   5-6
    finished application outline   5-7
    PBC   5-4
    profile configuration   5-5
Profile session   5-2
Profiler   5-2
program
    build   2-8
    run   2-8
project
    add DSP/BIOS configuration files   6-7
    adding files   2-4
    build   2-8
    create a   2-2
    Project Creation wizard   2-2
project configuration
    add new   2-11
    change active   2-10
    selecting   2-10