Code Composer Studio For Solaris

Getting Started Guide

Literature Number: SPRU556
February 2002

Printed on Recycled Paper

IMPORTANT NOTICE

Texas Instruments Incorporated and its subsidiaries (TI) reserve the right to make corrections, modifications, enhancements, improvements, and other changes to its products and services at any time and to discontinue any product or service without notice. Customers should obtain the latest relevant information before placing orders and should verify that such information is current and complete. All products are sold subject to TI's terms and conditions of sale supplied at the time of order acknowledgment.

TI warrants performance of its hardware products to the specifications applicable at the time of sale in accordance with TI's standard warranty. Testing and other quality control techniques are used to the extent TI deems necessary to support this warranty. Except where mandated by government requirements, testing of all parameters of each product is not necessarily performed.

TI assumes no liability for applications assistance or customer product design. Customers are responsible for their products and applications using TI components. To minimize the risks associated with customer products and applications, customers should provide adequate design and operating safeguards.

TI does not warrant or represent that any license, either express or implied, is granted under any TI patent right, copyright, mask work right, or other TI intellectual property right relating to any combination, machine, or process in which TI products or services are used. Information published by TI regarding third party products or services does not constitute a license from TI to use such products or services or a warranty or endorsement thereof. Use of such information may require a license from a third party under the patents or other intellectual property of that third party, or a license from TI under the patents or other intellectual property of TI.

Reproduction of information in TI data books or data sheets is permissible only if reproduction is without alteration and is accompanied by all associated warranties, conditions, limitations, and notices. Reproduction of this information with alteration is an unfair and deceptive business practice. TI is not responsible or liable for such altered documentation.

Resale of TI products or services with statements different from or beyond the parameters stated by TI for that product or service voids all express and any implied warranties for the associated TI product or service and is an unfair and deceptive business practice. TI is not responsible or liable for any such statements.

Mailing Address:

Texas Instruments
Post Office Box 655303
Dallas, Texas 75265

Preface

Read This First

About This Manual

To get started with Code Composer Studio ™ for Solaris 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 (CCStudio).

How to Use This Manual

This document contains the following chapters:

Chapter	Title	Description
1	Getting Started with Code Composer Studio for Solaris IDE	Walks you through the steps of setting up the CCStudio IDE and shows you how to access documentation.
2	Code Composer Studio for Solaris Code Generation Tools	Reviews development tools provided with CCStudio IDE.
3	Code Composer Studio for Solaris Debug Tools	Reviews the available debug tools and the device generations to which they apply.
4	Code Composer Studio for Solaris Optimization Tools	Provides information on profiling code.
5	Code Composer Studio for Solaris DSP/BIOS Kernel	Overview of the DSP/BIOS kernel featured in CCStudio IDE.
6	Code Composer Studio for Solaris Chip Support Library Overview	Provides and overview of the features and architecture of CSL.
7	Troubleshooting	Overview of troubleshooting responses.

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
```

- <install_dir> is the directory used to install Code Composer Studio for Solaris

Related Documentation From Texas Instruments

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

To access the online manuals:

Help→User 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

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

1 Getting Started with Code Composer Studio for Solaris IDE 1-1

Instructions on setting up the CCStudio IDE and tools that aid you in using CCStudio IDE, such as documentation.

1.1 Installation Instructions 1-2
System Requirements 1-2
License Agreement 1-2
Required Solaris Patches 1-3
Patch Installation 1-5
Installing Code Composer Studio for Solaris 1-5
User Setup 1-6
Co-existence of CCStudio for Solaris version 2.0 with version 1.1 1-8
Setting Paths and Environmental Variables for Command Line Tools 1-8
Printing from Code Composer Studio for Solaris 1-9
Uninstalling Code Composer Studio for Solaris 1-10
Installation of XDS510WS Emulation Support 1-10
1.2 Creating a System Configuration 1-11
1.3 Setting up CCStudio for Heterogeneous Debugging 1-14
1.4 UNIX Usability Enhancements 1-16
UNIX Environment Support 1-16
Remote Execution with Local Display 1-16
External Editor Support 1-17
Command Window Aliases 1-18
Printer Support 1-18
Side–by–side Installs and Multiple Instances: 1-18
Distributed Build Support 1-18
Path setting utility 1-19
1.5 UNIX Tips and Tricks 1-20
Specifying Source Directories 1-20
Customize GEL Toolbar 1-20
Starting Code Composer Studio for Solaris with a GEL file 1-21
Command Window Scripting 1-21
Specifying Workspace at Startup 1-22
1.6 Getting Started with CCStudio Tutorial 1-23
1.7 Accessing CCStudio Documentation 1-24

2 Code Composer Studio for Solaris Code Generation Tools 2-1
Code Composer Studio Code Generation Tools.

2.1 Code Generation Tools 2-2
2.2 Compiler Overview 2-3
Interfacing with Code Composer Studio 2-3
2.3 Assembly Language Development Tools 2-4
2.4 Assembler Overview 2-5
2.5 Linker Overview 2-6
2.6 Command Line Visual Linker 2-7
Running the vlnk Linker 2-7
2.7 C/C++ Code Development Tools 2-9

3 Code Composer Studio for Solaris Debug Tools 3-1
Reviews the available debug tools and the device generations to which they apply.

3.1 Overview of Applicable Debug Tools 3-2
3.2 Running the Program 3-3
3.3 Reviewing Your Source Code Using the Editor 3-4
External Editor 3-5
3.4 Introduction to Breakpoints 3-6
Software Breakpoints 3-7
Hardware Breakpoints 3-8
3.5 Watch Window 3-9
3.6 Probe Points 3-13
3.7 Simulator Analysis 3-17
3.8 Emulator Analysis 3-19
3.9 Displaying Graphs 3-20
3.10 Symbol Browser 3-22
3.11 General Extension Language (GEL) 3-23
3.12 Command Window 3-24
3.13 Pin Connect 3-25
3.14 Port Connect 3-26

4 Code Composer Studio for Solaris Optimization Tools 4-1
Reviews optimization tools provided with CCStudio IDE.

4.1 Profiler 4-2

5 Code Composer Studio for Solaris DSP/BIOS Kernel 5-1
Overview of the real-time components featured in CCStudio IDE.

5.1 DSP/BIOS Kernel Components 5-2
5.2 DSP/BIOS TextConf 5-3
5.3 DSP/BIOS Kernel 5-6

6 Code Composer Studio for Solaris Chip Support Library Overview 6-1
An overview of the features and architecture of the Chip Support Library.

6.1 Introduction to CSL 6-2
How the CSL Benefits You 6-2
6.2 CSL Configuration Files Generated by DSP/BIOS TextConf 6-3
Header file 6-3
Source file 6-4

7 Troubleshooting 7-1
Review troubleshooting guidelines.

7.1 Installation/Loading Code Composer Studio 7-2
7.2 File and Directory Names 7-6
7.3 General Debugging 7-6
7.4 Editor 7-7
7.5 General Extension Language – GEL 7-7
7.6 Graph Window 7-9
7.7 Miscellaneous 7-10

Figures

2–1 Code Development Flow 2-2
3–1 View Source Code 3-4
3–2 External Editor Icon 3-5
3–3 Watch Window Toolbar 3-9
3–4 Watch Window 3-9
3–5 Emulator Analysis Window 3-19
3–6 Symbol Browser Window 3-22
3–7 Command Window 3-24
5–1 File Generation by DSP/BIOS TextConf 5-3
5–2 Target Content Object Model (TCOM) 5-4
5–3 DSP/BIOS Modules 5-6
5–4 DSP/BIOS Execution Threads 5-7

Tables

1–1 Solaris 2.6/5.6/6 Required Patches 1-3
1–2 Graphics Card Patches for Solaris 2.6/5.6/6 1-3
1–3 Solaris 2.7/7 Required Patches 1-4
1–4 Graphics Card Patches for Solaris 2.7/7 1-4
1–5 Solaris 8 Required Patches 1-4
3–1 Debug Tools 3-2

Chapter 1

Getting Started with Code Composer Studio for Solaris IDE

This chapter applies to all platforms using Code Composer Studio™ for Solaris IDE (CCStudio IDE).

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

Topic **Page**

1.1	Installation Instructions	1-2
1.2	Creating A System Configuration	1-11
1.3	Setting up CCStudio for Heterogeneous Debugging	1-14
1.4	UNIX Usability Enhancements	1-16
1.5	UNIX Tips and Tricks	1-20
1.6	Getting Started with CCStudio Tutorial	1-23
1.7	Accessing CCStudio Documentation	1-24

1.1 Installation Instructions

These are the installation instructions you must follow to successfully install and use CCStudio.

System Requirements

Recommended configuration:

- ❑ Sun Ultra 5 300MHz workstation or better
- ❑ 256Mb RAM or more
- ❑ 800Mb free disk space
- ❑ Solaris 2.6/5.6/6 (January 1998 or later) or Solaris 2.7/7 (October 1998 or later) or Solaris 8 (January 2000 or later)

Minimum configuration:

- ❑ Sun Ultra 1 167MHz workstation or better
- ❑ 256Mb RAM or more
- ❑ 700Mb free disk space
- ❑ Solaris 2.6/5.6/6 (January 1998 or later) or Solaris 2.7/7 (October 1998 or later) or Solaris 8 (January 2000 or later)

To find out what your kernel revision is, type:

```
showrev –a | grep “Kernel version”
```

License Agreement

Please cd to the root directory of your product CD–ROM, and run the cc_license utility. When prompted, accept the license. Write down the displayed product key, as you will need it later for the O/S patch installation and for the product installation.

Required Solaris Patches

The following OS patches are essential for the proper functioning of CCStudio. Please contact your system administrator if necessary to ensure that these patches are installed on your machine. Installation of these patches requires root permissions.

Table 1–1. Solaris 2.6/5.6/6 Required Patches

Patch	Description
105633–32	X bug fixes for multi-threaded app support
105181–16	kernel thread problems fix
105591–06	Shared library patch for C++
106040–12	Fixes input/output problem
106125–08	Improvements for package install and remove (Optional)
106429–02	Supplement for patch 105181
105181–23	Patch for TConf (O/S kernel problems fix)
105210–33	Patch for TConf (LibC and watchmalloc fix)
105568–21	Patch for TConf (wait() and Thread sleep() fix)
105591–09	Patch for TConf (exception handling fix)

Patches for specific graphics cards (only those matching your cards are required).

Table 1–2. Graphics Card Patches for Solaris 2.6/5.6/6

Graphics Card	Graphics Card File	Patch
Elite3D	/dev/fbs/afb*	105361–10, 105363–24
GX	/dev/fbs/cgsix*	105492–02
Creator/ Creator3C	/dev/fbs/ffb*	105361–10, 150360–28
ZX	/dev/fbs/leo*	——
PGX	/dev/fbs/m640	105362–24
SX	/dev/fbs/sx*	——
TCX	/dev/fbs/tcx*	106391–01

Table 1–3. Solaris 2.7/7 Required Patches

Patches	Description
107078–18	Xserver not thread safe
106327–05	Shared library patch for C++
106541–17	Patch for TConf (libthreads fix)
106980–13	Patch for TConf (Java Client VM fix)
107544–03	Patch for TConf (java.awt.List fix)

Patches for specific graphics cards (only those matching your cards are required).

Table 1–4. Graphics Card Patches for Solaris 2.7/7

Graphics Card	Graphics card file	Patch
Elite3D	/dev/fbs/afb*	106114–12, 106148–12
GX	/dev/fbs/cgsix*	——
Creator/ Creator3D	/dev/fbs/ffb*	106145–12, 106148–12
PGX32	/dev/fbs/gfxp0*	107851–04
ZX	/dev/fbs/leo*	——
PGX	/dev/fbs/m640	106146–10
SX	/dev/fbs/sx*	——
TCS	/dev/fbs/tcx*	——

Table 1–5. Solaris 8 Required Patches

Patches	Description
108434–01	Shared library patch for C++
108921–07	Patch for TConf (FOCUS_LOST and FOCUS_GAINED events fix)
108940–12	Patch for TConf (core dump in some situations fix)

No patches for specific graphics cards required for Solaris 8.

Patch Installation

The required Solaris patches must be installed on each physical machine where CCStudio is to be run. To install the patches, follow these steps:

Step 1: cd to the Sun_OS_patches directory on the product CD.

Step 2: Log in as root.

Step 3: Run the O/S patch installation script with the license key obtained at the beginning of the installation procedure as a command line argument:

./os_patch_install.sh <license_key>

and follow the script directions. You will be prompted before the patches are installed.

Step 4: Reboot your machine when prompted by the script.

Installing Code Composer Studio for Solaris

Code Composer Studio may be installed on the user machine for a single user, or as a shared install accessible by multiple users.

- ❑ For the single user install, the user should complete steps described in this section, and then proceed to User Setup, found on page 1-6.
- ❑ For the shared install, administrator (root) should complete steps described in this section, and then every user that is going to use the shared install should complete steps described in User Setup, found on page 1-6.

Shared install saves disk space while allowing use of CCStudio by multiple users and/or on multiple workstations.

Step 1: Ensure that you have a supported OS version that has been properly patched (as described previously). This is essential for proper functionality of CCStudio for Solaris.

Step 2: Log in with your regular user ID if you are going to perform a single-user installation, or log in as root if you are going to perform installation for multiple users.

Step 3: If you have CCStudio for Solaris v1.1 installed on your machine, you can choose to continue using it alongside the new version, or to uninstall it. To uninstall, just remove the version 1.1 installation tree.

Step 4: If you have another release of CCStudio for Solaris v2.0 installed on your machine, uninstall it by removing the previous release installation tree.

Step 5: Install Code Composer Studio for Solaris as described below:

1) cd to the CCStudio directory on the product CD.

2) Run the product installation script with the license key as a command line argument:

   ./codecomposer–unpack.sh <license_key>

Note:

Do NOT run codecomposer–unpack.sh as a background task (with &) since the script requires input from the terminal.

3) Follow script directions.

4) If you do not have Adobe Acrobat Reader installed on your system, it is strongly recommended that you install it when prompted by the script. Adobe Acrobat Reader is needed to view online manuals provided with CCStudio for Solaris.

5) In case you are performing an installation for multiple users, ensure that the permissions in the CCStudio for Solaris installation hierarchy allow for read access to all files and directories, as well as executable access to directories, executable files and scripts for all users.

User Setup

Each user that is going to use CCStudio for Solaris version 2.0 must perform the following setup actions:

Step 1: Prepare for setup.

1) Log in with your regular user ID.

2) If the ${path} or ${PATH} settings in any of your resource files (.profile, .bash_profile, .bashrc, .cshrc, .tcshrc) reference a binary directory of any other release of CCStudio for Solaris version 2.0, remove those references.

3) If the ${path} or ${PATH} settings in any of your resource files (.profile, .bash_profile, .bashrc, .cshrc, .tcshrc) reference a binary directory of CCStudio for Solaris version 1.1, and you are not going to use that version any more, then remove those references.

4) If Adobe Acrobat Reader is installed on your computer but is not in your user path, determine and write down the location of acroread binary. For example, if Adobe Acrobat Reader 4 is installed in /home/johnsmith/Acrobat4, the binary will be located in /home/johnsmith/Acrobat4/bin.

Step 2: Run script <install_dir>/ti/solaris/cc/bin/cc_setup and follow the script directions.

Note:

Do NOT run cc_setup in the background (with &) at this point, since the script requires input from the terminal. Subsequent launches of cc_setup by the same user can be performed with & as no terminal input will be required.

The script will:

- Create the configuration directories and files inside the ${HOME}/.CodeComposer directory tree;
- Add the CCStudio for Solaris version 2.0 binary directory to the PATH settings in your shell resource files (.profile, .bash_profile, .bashrc, .cshrc, .tcshrc). This modification will not be applied if the existing PATH settings include CCStudio for Solaris version 1.1 binary directory;
- Add the Adobe Acrobat Reader binary directory to the PATH settings in your shell resource files, unless it is already in your PATH;
- Modify your .Xdefaults file, if necessary.

Old versions of the modified files will be backed up as <filename>.ccs.bak (e.g. .profile.ccs.bak).

If prompted by the script, log out from the machine and then log back in with your regular user ID.

Step 3: The CCStudio for Solaris version 2.0 binary directory will now be in your path, unless CCStudio for Solaris version 1.1 binary had been in your path before the setup started. In the latter case, refer to Coexistence of CCStudio for Solaris version 2.0 with version 1.1, found below.

Type “cc_setup” in the command prompt. When the CCStudio setup window pops up, set up a system configuration with a simulated board as described in Creating a System Configuration, page 1-11.

You are now ready to start using CCStudio for Solaris version 2.0 with simulation capability. To start CCStudio for Solaris, type ”cc_app” in the command prompt.

Co-existence of CCStudio for Solaris version 2.0 with version 1.1

CCStudio v2.0 and CCStudio v1.1 may coexist if desired, but only one of them can run at a time. In addition, CCStudio binaries and tools have the same name between different versions of CCStudio. Therefore, scripts should be set up to switch the active path between the different installs. For example, the set_CCS1.1_c6000_install and set_CCS2.0_install scripts provided in <install_dir>/ti/solaris/cc/bin can be used to switch between the two installations. The user should copy these scripts to the home directory and modify them to match his/her directory layout and interactive shell. Note that the scripts should be sourced to take effect, e.g.:

> source set_CCS1.1_c6000_install

Setting Paths and Environmental Variables for Command Line Tools

Code Generation and DSP/BIOS tools provided with the CCStudio for Solaris work on the command line. To automatically set paths and environmental variables for these tools, you can use scripts setpath.csh and setpath.sh provided in <install_dir>/ti/solaris/cc/bin. To update settings in your current command shell, execute:

source <install_dir>/ti/solaris/cc/bin/setpath.csh (if you use a C shell)

or

. <install_dir>/ti/solaris/cc/bin/setpath.sh (if you use the Bourne or Bash shell)

If you want to automatically update these settings in a new command shell, place the above line(s) in the appropriate shell resource file. However, note that some executables installed with CCStudio for Solaris (e.g. make) may conflict with regular system commands employing identical names.

Printing from Code Composer Studio for Solaris

At this point, you should be able to use printer(s) installed on your workstation or your local network to print directly from CCStudio for Solaris, without any additional setup.

By default, CCStudio uses the Unix command lp –c –s <filename> to process print requests. If you have problems printing from CCStudio, check that the above command works from the Unix command prompt.

If you wish to modify the print command used by CCStudio, or to enable CCStudio to use multiple printers, run utility cc_printcfg.sh located in the directory ti/solaris/cc/bin:

Step 1: When prompted, enter “8” to display the current printer configuration.

Step 2: To create a new printer command, enter “1” and add a new printer spool definition. A printer spool definition is a print command that would normally be executed from the Unix shell to print a file on the desired printer, followed by the string “%s” enclosed in double quotes. “%s” should be present in any spool definition. An example of the printer spool definition is:

lp –c –s “%s”

Step 3: After setting the new printer spool definition, enter “4” to add a new printer nickname. An arbitrary printer nickname can be chosen, and that name will appear in all CCStudio for Solaris print dialogs.

Note:

The cc_printcfg.sh utility does not install Unix printers. Contact your system administrator for that purpose.

Uninstalling Code Composer Studio for Solaris

To uninstall your Code Composer Studio installation, complete the following steps:

Step 1: Remove the Code Composer Studio directory.

Step 2: If you are not using Code Composer Studio v1.1, remove the .Code-Composer directory.

If you want to continue using Code Composer Studio v1.1, cd to the ${HOME}/.CodeComposer directory and remove all subdirectories whose name matches patterns “v2*” and “config.*”.

Step 3: Remove the Code Composer Studio bin directory from your PATH.

Installation of XDS510WS Emulation Support

The XDS510WS device provides processor emulation capability. It connects to the SCSI port on your workstation. It also connects through a JTAG connector to an application board. CCStudio for Solaris is shipped with emulation drivers for use with the specific processors identified in the product release notes. To make use of emulation with CCStudio for Solaris, you must purchase and install the XDS50WS emulator and bundled device drivers. If you have purchased this package, you can install it now.

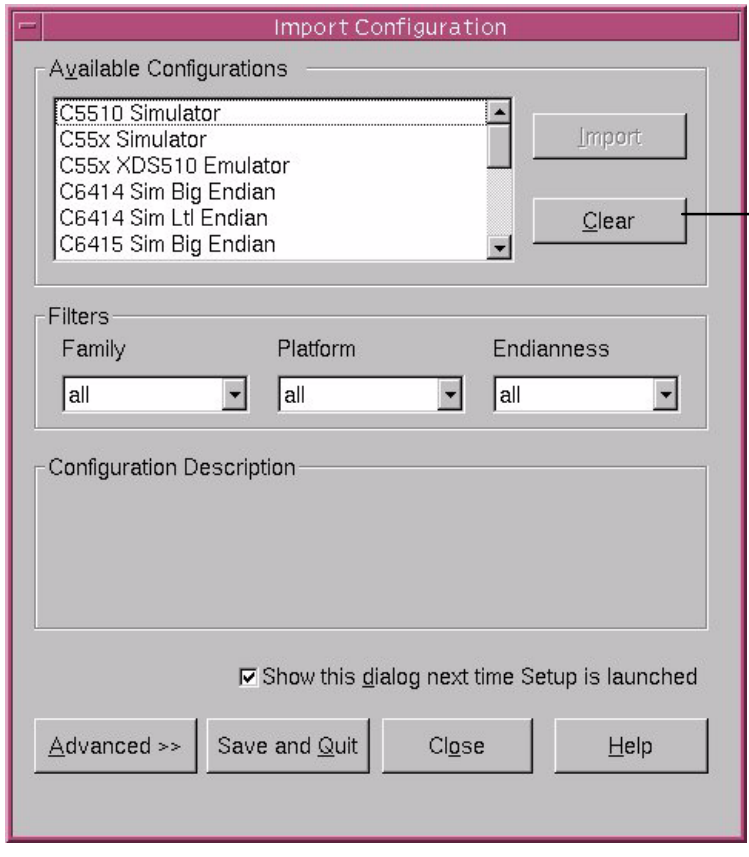
1.2 Creating a System Configuration

CCStudio Setup allows you to configure the CCStudio IDE software to work with different hardware or simulator targets. It also 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 CCStudio product.)

To create a system configuration using a standard configuration file:

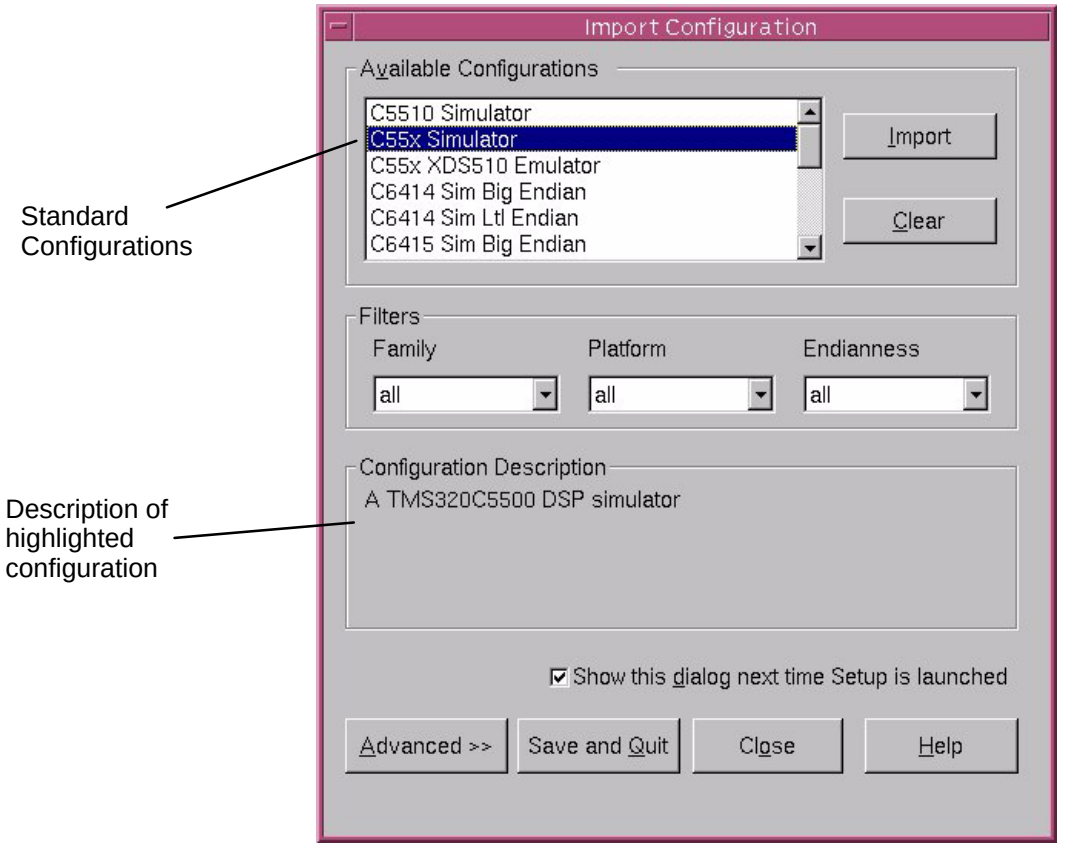
Step 1: Start CCStudio Setup by executing the cc_setup command.

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



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 CCStudio product).

Step 5: Click the Import button to import your selection to the system configuration currently being created in the CCStudio 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: If your system configuration contains simulated boards only, you can click the Yes button to start Code Composer Studio for Solaris when you exit CCStudio Setup. This will automatically open the CCStudio IDE using the configuration you just created.

If your system configuration contains any hardware targets, click the No button when asked whether to start Code Composer Studio for Solaris.

Step 8: If your system contains hardware targets, and you are using the XDS510WS device to connect to them, load the appropriate XDS510WS device driver by typing in the command prompt:

load_XDS510ws_c55x (for TMS320C55x targets)

or

load_XDS510ws_c64x (for TMS320C64x targets)

Note:

If your PATH has not been updated to include the CCStudio bin directory, launch the above scripts from directory <install_dir>/ti/solaris/cc/bin.

You can now start CCStudio for Solaris by typing ”cc_app” in the command prompt.

1.3 Setting up CCStudio for Heterogeneous Debugging

Code Composer Studio for Solaris can be configured to perform heterogeneous debugging. For more information on this topic, please see the online help provided with CCStudio for Solaris.

Step 1: Install the XDS emulator and connect the host JTAG cable to the heterogeneous target.

Step 2: Start Code Composer Studio Setup.

Step 3: Click Clear and then click Yes to clear the System Configuration pane found in the Import Configuration dialog box.

Step 4: Click Close to exit the Import Configuration dialog box.

Step 5: Drag the Heterogeneous Multi-Target (Texas Instruments) driver from the Available Board/Simulator Types pane to the System Configuration pane.

If the heterogeneous device driver is not in the Available Board/Simulator Types pane, scroll to find “Heterogeneous Emulator” in the list of available configurations. Select Heterogeneous Emulator and click import. Save the configuration and close CCSetup without starting CCStudio. Then, open CCSetup and select Heterogeneous Multi-Target.

Step 6: In the Board Properties dialog box, click Next to proceed to the Board Properties tab.

Step 7: Enter the SCSI port address of the XDS510WS emulator box.

Step 8: Click Next to proceed to the Processor Configuration tab.

Step 9: Select the CPU types required for your target system and add them to the list of processors.

The processors must be selected in the order that the devices appear on the scan path (from TDI to TDO).

Step 10: Click Finish to close the Board Properties dialog and complete the driver setup process.

Step 11: Select File→Exit, and then click Yes to save system configuration changes and exit CCSetup. When asked whether to start Code Composer Studio for Solaris, click the No button.

Step 12: If you are using the XDS510WS device to connect to hardware targets, load the appropriate XDS510WS device driver by typing in the command prompt:

load_XDS510ws_c55x (for TMS320C55x targets)

or

load_XDS510ws_c64x (for TMS320C64x targets)

Note:

If your PATH has not been updated to include the CCStudio bin directory, launch the above scripts from directory <install_dir>/ti/solaris/cc/bin.

You can now start CCStudio for Solaris by typing ”cc_app” in the command prompt.

1.4 UNIX Usability Enhancements

The enhancements to Code Composer Studio for Solaris that are listed in this chapter provide better product integration in the UNIX environment. These enhancements give you access to tools and commands commonly used in the UNIX environment.

UNIX Environment Support

CCStudio for Solaris is integrated into the X Windows environment. The Common Desktop Environment (CDE), the default window manager shipped with Solaris, is the supported window manager. CCStudio for Solaris has functionality similar to other X Windows GUIs. This functionality includes window manager integration, focus policy, color policy, as well as support for X11 clipboard cut and paste.

Remote Execution with Local Display

CCStudio for Solaris can be run remotely with local display as expected of any X Windows application.

Running CCStudio for Solaris remotely with a Solaris, HP–UX, or Linux local display.

Depending on your system configuration, it may be necessary in some cases to modify the Unix environment to support this. Specifically, the local machine must allow X client connections from the remote machine. This can usually be done with the xhost command. For example, from the local machine, run the following command:

```
xhost +remote_machine
```

The `$DISPLAY` environment variable must also be set on the remote machine in order to display the windows on the local machine. After logging onto the remote machine you may have to set this environment variable.

For example, if the user is logged onto the remote machine, and running a C shell, the command would be:

```
setenv DISPLAY local_machine:0.0
```

If running a sh–based shell, an equivalent command is

```
DISPLAY=local_machine:0.0
export DISPLAY
```

For further information consult a UNIX reference book, guru or the man pages.

Running CCStudio for Solaris remotely with a Windows local display using Exceed™.

The sample procedure to use Exceed 6.0.1 is as follows:

Step 1: Select Start–> Programs–> Exceed–> Xstart.

Step 2: In the Xstart GUI, select:

```
Start Method = REXEC (TCP/IP)
Program Type = X Windows
Host Type = SUN
Command = @(XTerm, method = stdappdb) –display
@d&;
Prompt = (None)
```

Step 3: Enter the User ID, Password, and Host name in the corresponding editboxes.

Step 4: Press the “Run!” button in the menu bar. Wait until an xterm window with a Unix prompt opens on the Windows desktop.

Step 5: Check whether the DISPLAY variable is set in the Unix shell and points to the Windows workstation, by executing command

```
echo $DISPLAY
```

If the variable is not set correctly, set it as described in the previous section.

Step 6: Run cc_setup and then cc_app.

If you experience difficulties using Exceed™, please contact the Exceed™ vendor (Hummingbird corp.).

External Editor Support

CCStudio allows you to use your preferred UNIX editor, in place of the CCStudio integrated editor, to edit files. However, the CCStudio integreated editor must be used to debug programs.

In order to use console-based editors such as vi, a new xterm must be invoked. Enter the full path to xterm in the Path to Editor box, and then enter ‘–e vi’ as an argument in the “Arguments to Apply” box. Next, place ‘+’ in the “Open file at a specific line” box, and make sure the “Apply arguments after the filename” checkbox is not selected.

To use xemacs, replace the contents of the Path to Editor box with the full path to xemacs, and place '+' in the "Open file at a specific line" box. No other arguments are required.

Note:

If you are using a console-based editor in an xterm, the "Apply arguments after filename" checkbox must be unchecked.

Command Window Aliases

The Command Window supports several DOS-like commands used to retrieve system information. CCStudio for Solaris provides Unix aliases to the DOS-like commands supported by the Command Window. Specifically, "pwd" is equivalent to "cd" while "ls" is equivalent to "dir". These aliases allows you to use these common UNIX commands in the Command Window. However, these aliases are not available in Code Composer Studio for Windows and should not be used in Command Window scripts to maintain portability with CCStudio for Windows.

Printer Support

As part of the integration of CCStudio for Solaris into the UNIX environment, printing in Code Composer Studio for Solaris invokes the UNIX utility 'lp'. The UNIX printer settings can be set with the cc_printcfg utility.

Side–by–side Installs and Multiple Instances:

CCStudio for Solaris supports side–by–side installs of version 2.0 with version 1.1 on a single machine, as well as overinstalling an old version with a newer version. Multiple users can each run one instance of CCStudio on a single Solaris machine, using one shared install if desired. In addition, running multiple instances of CCStudio by the same user on a single machine is supported for co-verification purposes only.

Distributed Build Support

CCStudio for Solaris supports distributed builds using an external make utility. Your build environment and make utility must already support distributed building to utilize this ability when compiling a Code Composer Program. CCStudio for Solaris does not enable distributed build support.

Path setting utility

A utility has been added for automatic setting of environmental variables and paths used by the Code Generation tools. This utility is provided to set up a convenient access to all the command line tools from a command shell. To update setting in your current command shell, execute:

source <install_dir>/ti/solaris/cc/bin/setpath.csh (in case you use a C shell)

or

. <install_dir>/ti/solaris/cc/bin/setpath.sh (in case you use the Bourne or Bash shell)

If you wish to automatically update these settings in a new command shell, place the above line(s) in the appropriate shell resource file. However, note that some executables installed with CCStudio for Solaris (e.g. make) may conflict with regular system commands employing identical names.

1.5 UNIX Tips and Tricks

The following are tips for automating Code Composer Studio for Solaris to better integrate into the UNIX paradigm. The automations can be used to have Code Composer Studio automatically start certain processing upon startup.

Specifying Source Directories

Code Composer Studio for Solaris v2.0 does not support project management. Consequently, the user is prompted to browse for source files when they are not located in the same directory as the related binary. The user can specify directories to be searched by adding directories through the option menu. For more information, see topic Defining a Search Path for Source Files provided in the online help: Help→Contents→Using CCStudio for Solaris IDE→User Interface→Customizing the User Interface. Alternately, the user can add directories to the source path by invoking the GEL function GEL_SrcDirAdd(). The following code illustrates this example:

```
GEL_SrcDirAdd(”/project/source/”);
```

These GEL commands can also be added to GEL files to automatically create a source directory tree. It should be noted that GEL_SrcDirAdd() cannot be used in GEL files that are loaded at startup.

Customize GEL Toolbar

A gel file can be loaded that adds functions to the toolbar. Using this feature, common UNIX commands and tools can be added to the toolbar. The following snippet of a GEL file adds an xterm and xemacs to the menu bar.

```
menuitem ”My Functions”;
hotmenu Xterm()
{GEL_System(”xterm”);}
hotmenu xemacs()
{GEL_System(”xemacs”);}
```

For more information, see topic Adding GEL Functions to the Menu provided in the online help: Help→Content→ Using CCStudio for Solaris IDE→General Extension Language→Working with GEL Functions and Files.

Starting Code Composer Studio for Solaris with a GEL file

Code Composer Studio can be started with a GEL file on the command line. It should be noted that if a GEL file is specified for the processor in setup, then that file will supercede the GEL file given on the command line. When GEL files are loaded, the StartUp() function, if present will automatically run. This function can be used to load and run target code. The following example GEL function loads a program, sets some breakpoints and animates.

```
StartUp(){
   GEL_Load(”/home/test/modem/Modem.out”);
   GEL_Go(main);
   GEL_BreakPtAdd(0x000717);
   GEL_BreakPtAdd(0x000722);
   GEL_BreakPtAdd(0x0005bb);
   GEL_BreakPtAdd(0x0005e6);
   GEL_BreakPtAdd(0x000623);
   GEL_BreakPtAdd(0x00059a);
   GEL_BreakPtAdd(0x000591);
   GEL_Animate();
}
```

For more information, see topic Autoexecuting GEL Functions upon Startup provided in the online help: Help→Contents→Using CCStudio for Solaris IDE→General Extension Language→Working with GEL Functions and Files.

Command Window Scripting

The CCStudio Command Window can be a powerful scripting tool for automation of testing and debugging on Code Composer Studio. Information on the Command Window commands can be found in the online help topic Help→Contents→Command Window. Command Window scripts can be run using the ”take” command. Furthermore, ”eval” can be used to evaluate any C expression, including GEL functions. This adds further functionality to the Command Window, allowing the user to create GEL functions to perform any function not available through the Command Window commands.

While Command window scripts cannot be specified on the command line, ”init.cmd” is the default script that is run when the command window is first open. The ‘init.cmd’ file must be in the current working directory. The following Command Window commands could be placed within init.cmd to automatically load and a program and run to main when the Command Window is launched.

```
load /home/test/Modem.out
reset
restart

br
ba main
run                                          run to main
```

Specifying Workspace at Startup

Code Composer Studio normally starts with a blank workspace. However, a workspace can be loaded from within Code Composer Studio or specified at startup. Code Composer Studio will accept a workspace file as a command line parameter. For example:

```
cc_app test.wks
```

For more information on using workspaces, see topic Workspace Basics in online help: Help→Contents→Using CCStudio for Solaris IDE.

Workspaces save many elements including loaded GEL files and open windows. This allows the user to save a workspace that has the Command Window present. If Code Composer Studio is then run specifying that workspace, the Command Window will be automatically open when Code Composer Studio is launched and the ”init.cmd” script will run if it exists. This facility can be used to automatically start programs that perform various commands.

Similarly, if a workspace is saved that has an open GEL file, when that workspace is later loaded, either from Code Composer Studio or from the command line, the GEL file will also be loaded. The StartUp() function if it exists is run when the GEL file is loaded, this is a second method for specifying GEL files at startup.

The combination of Command Window scripts and GEL files available to the user in a workspace provides a powerful and versatile method for the user to automate the running and testing of programs with Code Composer Studio. Once the scripts, GEL files and workspaces are created, they can be used to automatically launch Code Composer Studio with specific instructions and tests for various scenarios.

1.6 Getting Started with CCStudio Tutorial

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

To access the CCStudio Tutorial, follow these steps:

1) Start the CCStudio IDE.

2) From the CCStudio Help menu, select Tutorial.

1.7 Accessing CCStudio Documentation

The CCStudio online help provides access to platform-specific documentation. The list of features provided with CCStudio can be found in the Release Notes or in the Welcome section of the online help.

To open the CCStudio Help, select Help→Contents. The Welcome page of the CCStudio Help displays. The table below lists the links on the Welcome page that provide important information:

For this information...	click this link.
CCStudio 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 CCStudio interface:

- ❏ To obtain the description of an instruction or register while editing your source code in a CCStudio 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 CCStudio 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.

Chapter 2

Code Composer Studio for Solaris Code Generation Tools

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

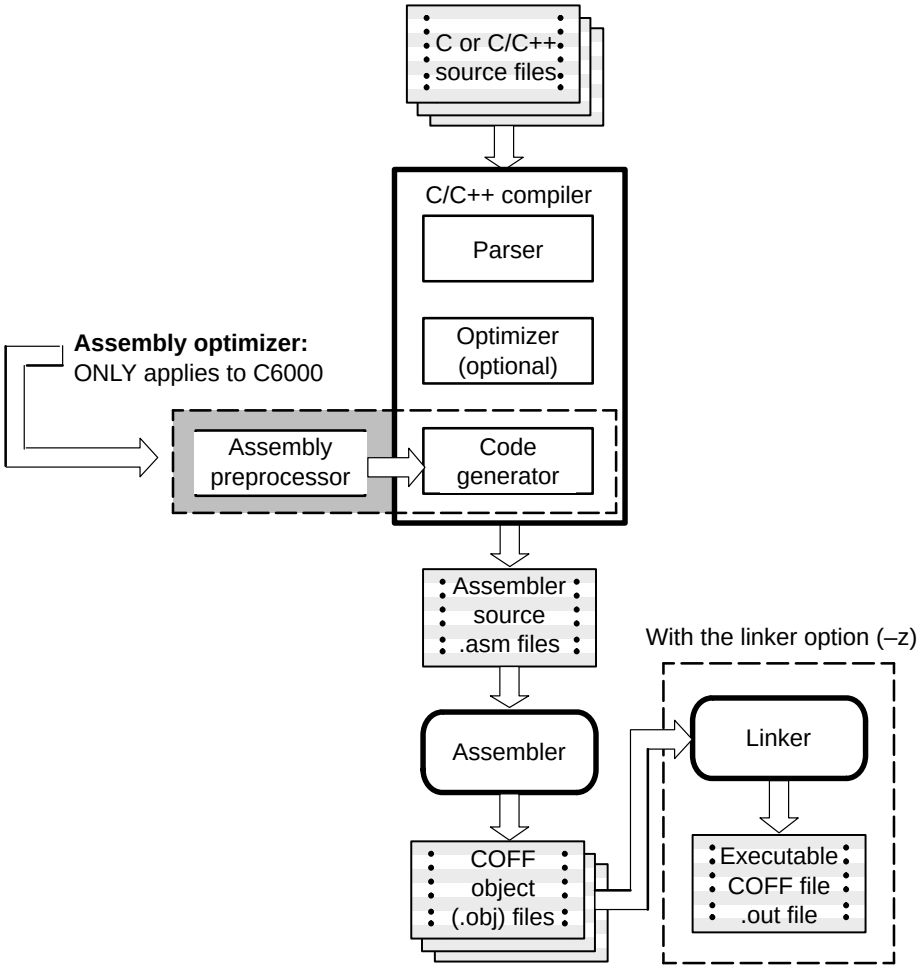
The different platforms offered with the CCStudio IDE are supported by a set of software development tools. These tools include an optimizing C/C++ compiler, an assembler, a linker, and assorted utilities. This chapter discusses these tools and shows you how to use them.

Topic **Page**

2.1 Code Generation Tools 2-2
2.2 Compiler Overview 2-3
2.3 Assembly Language Development Tools 2-4
2.4 Assembler Overview 2-5
2.5 Linker Overview 2-6
2.6 Command Line Visual Linker 2-7
2.7 C/C++ Code Development Tools 2-9

2.1 Code Generation Tools

Figure 2–1. Code Development Flow



2.2 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.

2.3 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.

2.4 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

2.5 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

2.6 Command Line Visual Linker

The vlnk linker is a command-line linker provided with your UNIX version of Code Composer Studio.

Running the vlnk Linker

The command line syntax for vlnk is:

vlnkxxx [–option|filename1 ... –option|filenameN]

You can arrange options and filenames in any order.

vlnkxxx	is the command that invokes the vlnk tool. This command is target-specific. For example, the C54x invocation is vlnk500.
options	are the options supported by the legacy TI linker, as well as some additional options for the vlnk tool.
filename	A file specification can include an absolute or relative path, and may be surrounded by double quotes. When specifying a relative path, the starting point is the current working directory. Use the –l option to specify files that are to be found via a search path. The file(s) can be one of the following:

Extension	**Description**
.rcp, .rdq	Visual Linker recipe file. (You can create a recipe file if you have the PC version of the Visual Linker.) If you specify a recipe, you cannot specify a linker command file.
.obj, .o, .out	COFF object file
.lib	library of COFF object files
.cmd, .lcf, .lnk, .arg, .lkv	legacy linker command file. If you specify a command file, you cannot specify a recipe file.
.mem	vlnk tool memory description
other	The first bytes in the file are used to classify it as .obj, .lib, or .cmd

If vlnk is invoked with no options or files, help information is displayed.

There are two ways to use the vlnk command:

- **In the same way that you use the legacy TI linker.**

  The vlnk command is compatible with the current TI linker. You can invoke vlnk with the same options, object files, and command file that you use with the TI linker.

  For example:

  ```
  vlnk hello.obj –lrts.lib link.cmd –o hello.out –m
  hello.map
  ```

  links hello.obj with the necessary support from the runtime library rts.lib. The runtime library is found using a search path. The link uses the memory description and allocation commands contained in lnk.cmd. The executable is named hello.out, and the memory layout is found in hello.map.

  In this example:

  ```
  vlnk phone.cmd –dead –vectors .vectors
  ```

  the link uses the options, files, memory description, and allocation contained in phone.cmd. But it also uses the vlnk tool features of dead-code elimination (–dead) to reduce memory usage. This command line example also specifies the section that holds the interrupt vectors (–vectors .vectors) to insure that interrupt code is not eliminated.

- **With a previously created recipe file**

  If you have the PC version of the Visual Linker, you can create a recipe file (.rcp). When you invoke vlnk with a recipe file, you can also specify object files, libraries, and linker options (except for –a, –ar, –r, –e, –vectors, and –dead) on the command line. You cannot specify a legacy linker command file.

  Command line arguments always override and overwrite the arguments in the recipe's .arg file. (An .arg file is created when a recipe is created.) If no additional arguments are specified, the contents of the .arg file are used and not overwritten.

  For example:

  ```
  vlnk modem.rcp
  ```

  performs a link using the options, files, memory description, and allocation contained in modem.rcp.

  This example:

  ```
  vlnk modem.rcp –o test1.out
  ```

  performs the link as in the previous example, but creates an output file with the name of test1.out. The "–o test1.out" specification replaces the existing arguments in modem.rcp.

2.7 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
- **TMS320C55x** devices:
  - An instruction-accurate software simulator
  - An extended development system (XDS510™) emulator

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

Chapter 3

Code Composer Studio for Solaris Debug Tools

This chapter applies to all platforms using Code Composer Studio™ (CCStudio) 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 the CCStudio IDE.*

The CCStudio 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.

Topic **Page**

3.1 Overview of Applicable Debug Tools 3-2
3.2 Running the Program 3-3
3.3 Reviewing Your Source Code Using the Editor 3-4
3.4 Introduction to Breakpoints 3-6
3.5 Watch Window 3-9
3.6 Probe Points 3-13
3.7 Simulator Analysis 3-17
3.8 Emulator Analysis 3-19
3.9 Displaying Graphs 3-20
3.10 Symbol Browser 3-22
3.11 General Extension Language (GEL) 3-23
3.12 Command Window 3-24
3.13 Pin Connect 3-25
3.14 Port Connect 3-26

3.1 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 CCStudio IDE.

Table 3–1. Debug Tools

Debug Tools	C64x	C55x
Breakpoints	✓	✓
Watch Window	✓	✓
Probe Points	✓	✓
Simulator Analysis	✓	
Emulator Analysis (Called Advanced Features for ARM)		✓
Graphs	✓	✓
Symbol Browser	✓	✓
GEL	✓	✓
Command Window	✓	✓
Pin Connect	✓	✓
Port Connect	✓	✓

3.2 Running the Program

To run a program, follow these steps:

Step 1: Choose File→Load Program.

Select the program you want to run (the .out file), and click Open.

The CCStudio 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 CCStudio IDE also automatically opens a tabbed area at the bottom of the window to show the output that the program sends to stdout.)

Step 2: Choose View→Mixed Source/ASM.

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

Step 3: 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 CCStudio IDE searches for help on that instruction. This is a good way to get help on an unfamiliar assembly instruction.

Step 4: Choose Debug→Go Main to begin execution from the main function. The execution halts at main and is identified by  .

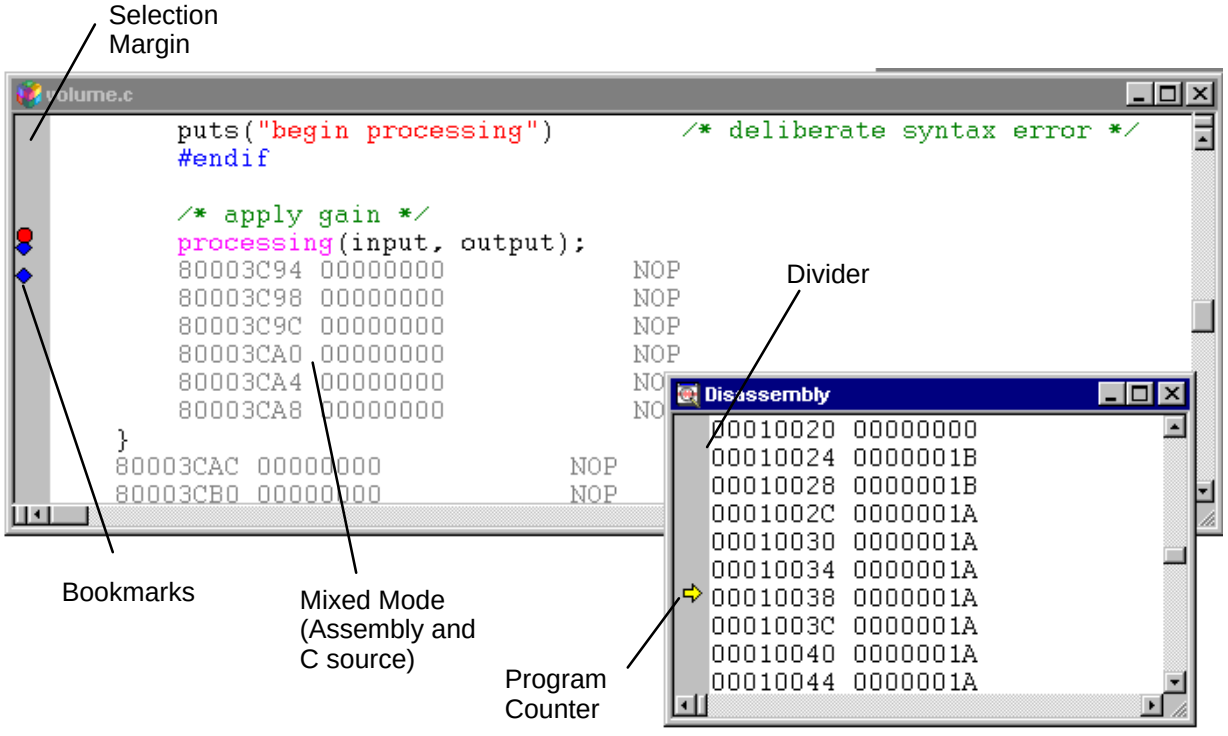
Step 5: Choose Debug→Run or click the  (Run) toolbar button to run the program.

Step 6: Choose Debug→Halt to quit running the program.

3.3 Reviewing Your Source Code Using the Editor

Open the file *filename*.c by selecting File→Open to display the source code.

Figure 3–1. 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. Keywords, 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).

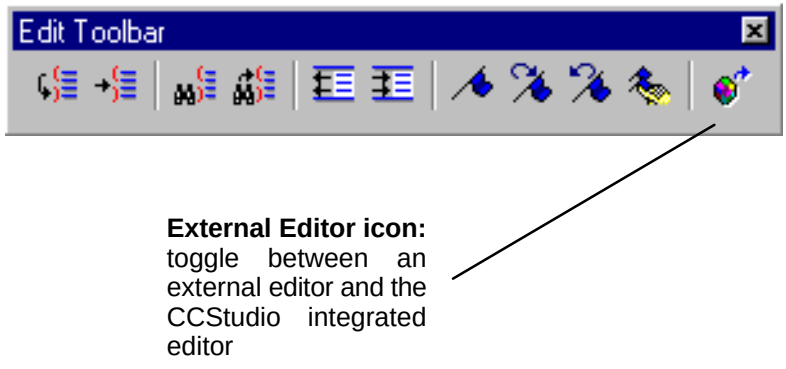
- **Keyboard Shortcuts.** The default keyboard shortcuts can be changed and new keyboard shortcuts can be created for any editing or debugging commands that can be invoked from a document window. Keyboard shortcuts can be modified through the customize dialog box in the Options menu.
- **Bookmarks.** Use bookmarks to find an maintain key locations within your source files. A bookmark can be set on any line of any source file.

External Editor

The CCStudio 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 CCStudio integrated editor must be used to debug your program.

Figure 3–2. External Editor Icon



3.4 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 disassembly 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:

CCStudio IDE tries to relocate a breakpoint to a valid line in your source window. CCStudio IDE places a breakpoint icon in the selection margin beside the lineon which it locates the breakpoint. If CCStudio 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 3.7, Simulator Analysis, on page 3-17, and section 3.8, Emulator Analysis, on page 3-19.

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.

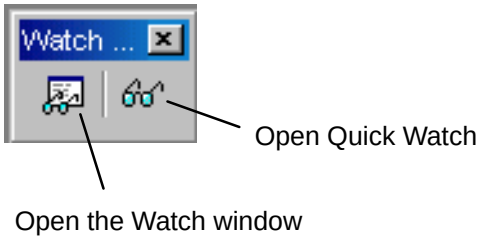
3.5 Watch Window

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.

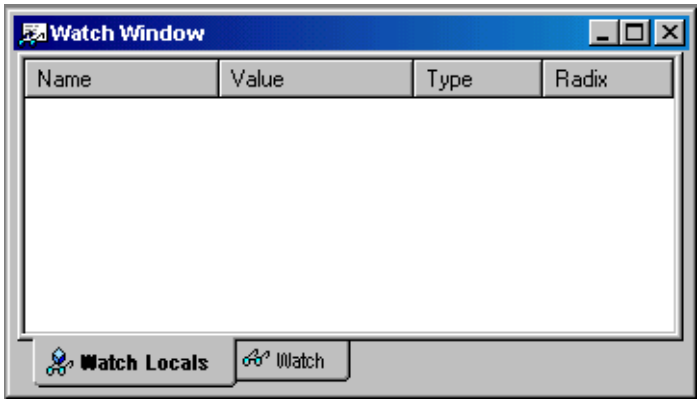
Figure 3–3. Watch Window Toolbar



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 3–4. 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 breakpoints 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.

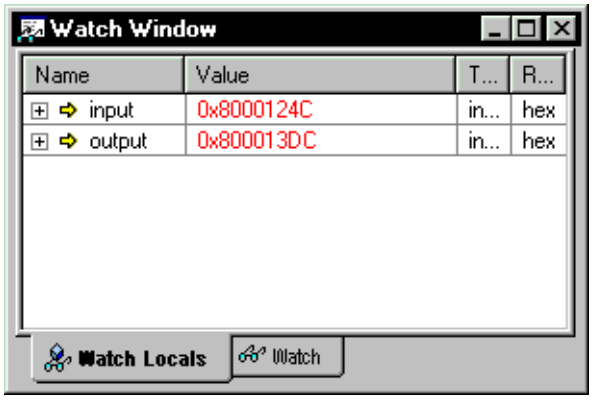
The selection margin indicates that a breakpoint has been set (red icon). If you disable the selection margin (Options→Customize) the line is highlighted in magenta.

Step 5: Choose View→Watch Window.

A separate area in the lower–right corner of the CCStudio 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.

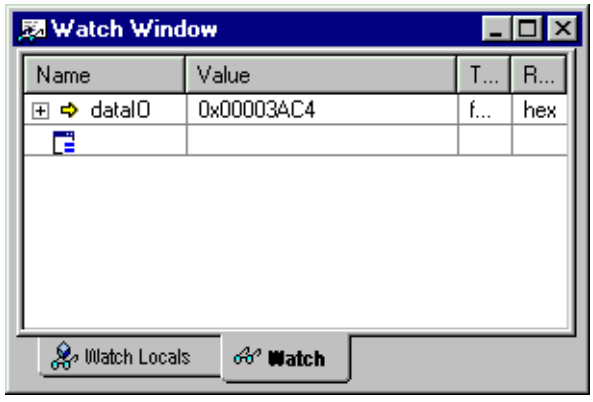
Step 7: Choose Debug→Run, or press F5, or press the  Icon.



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)

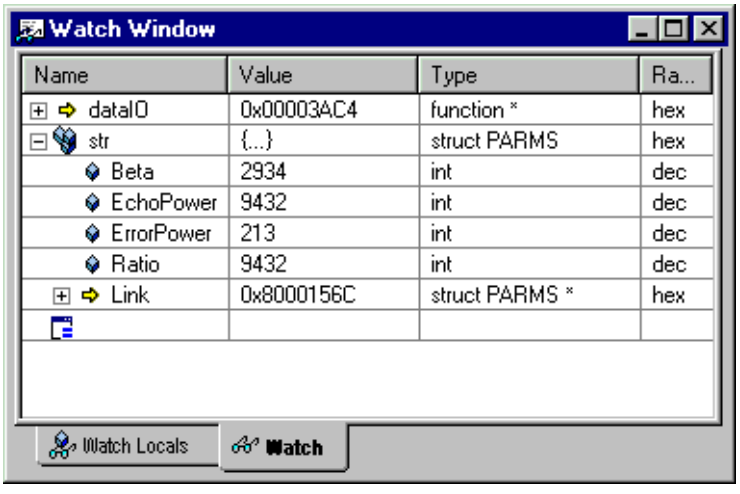
In addition to watching the value of a simple variable, you can watch the values of the elements of a structure.

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 CCStudio IDE expands this line to list all the elements of the structure and their values. (The address shown for Link may vary.)



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.

3.6 Probe Points

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

- ❏ transfer input data from a file on the host workstation 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 workstation 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 workstation 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.

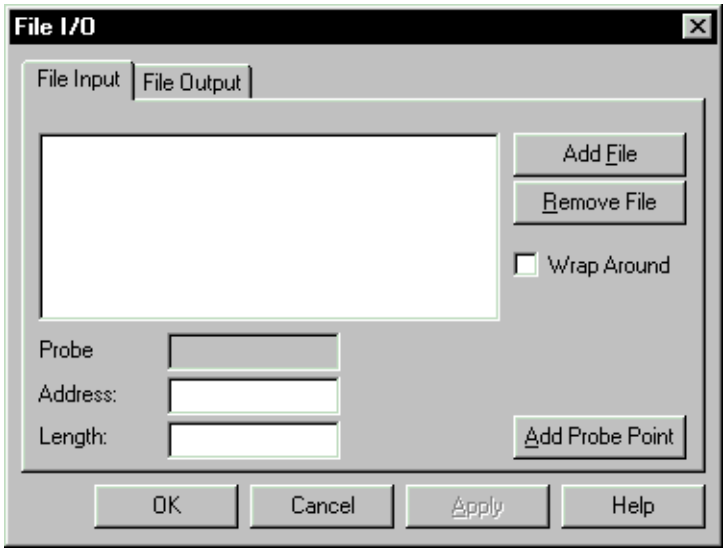
Step 1: Choose File→Load Program. Select *filename*.out, and click Open.

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.

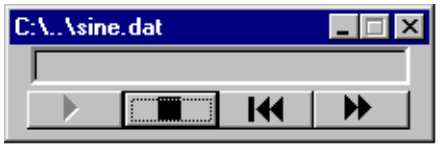
Step 5: From the File menu, choose File I/O. The File I/O dialog appears so that you can select input and output files.



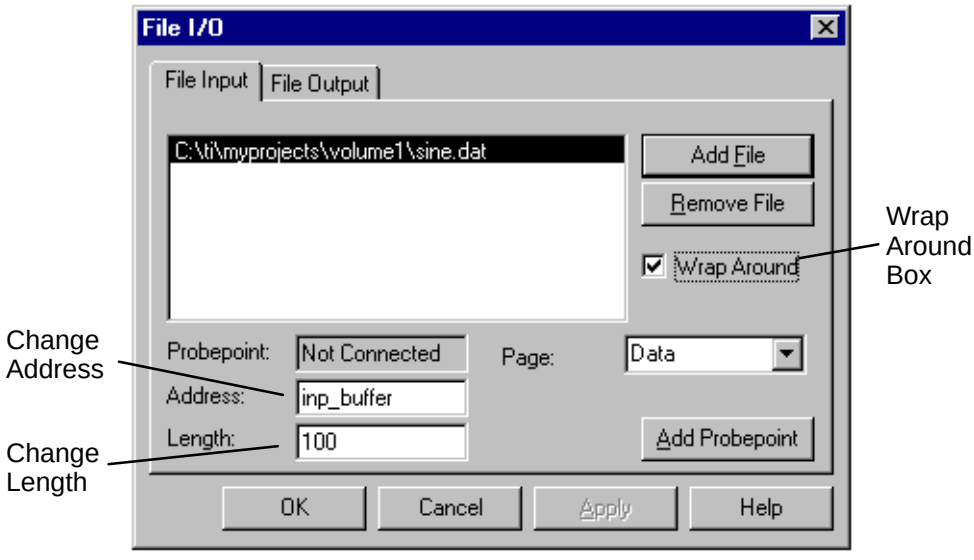
Step 6: In the File Input tab, click Add File.

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.

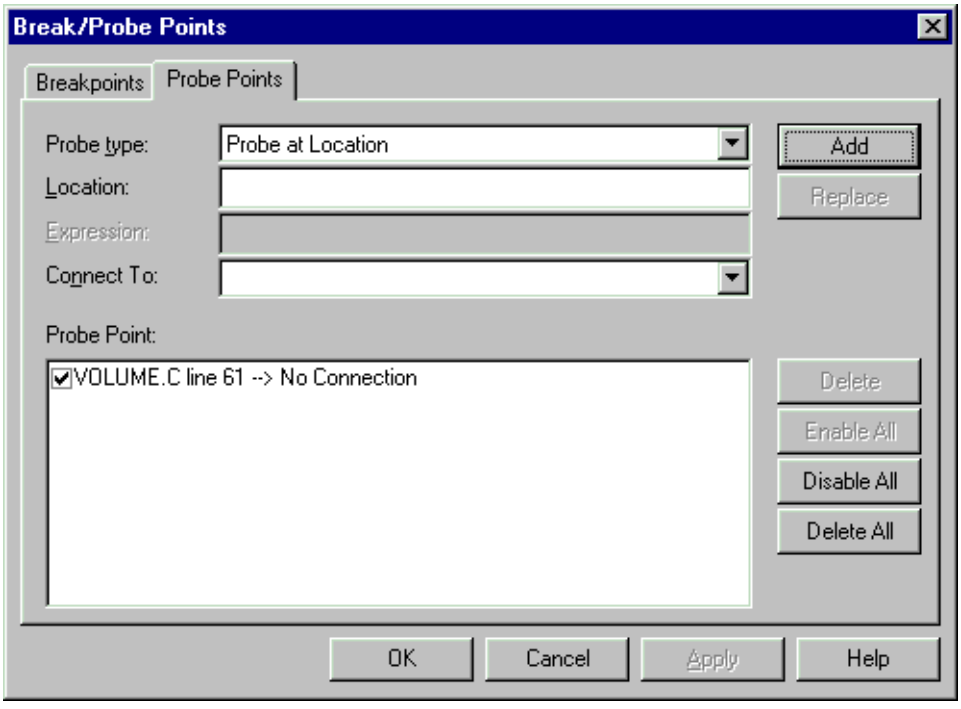


Step 8: In the File I/O dialog, change the Address and the Length values. Also, put a check mark in the Wrap Around box.



- 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 CCStudio 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.



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.

3.7 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.

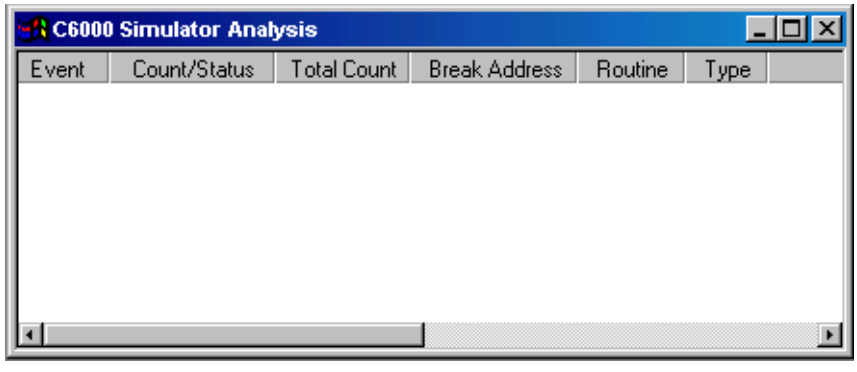
User Options:

- ❑ 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

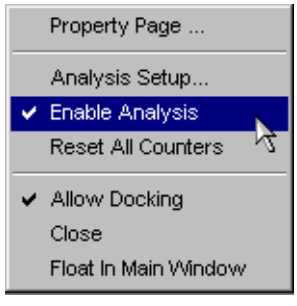
To use the Simulator Analysis tool:

Step 1: Load your program using the CCStudio 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 CCStudio IDE to run or step through your program.

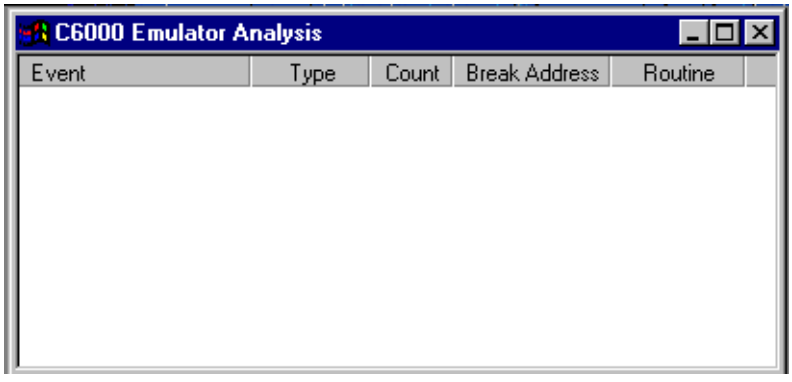
Step 6: Analyze the output of the analysis tool.

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

3.8 Emulator Analysis

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

Figure 3–5. Emulator Analysis Window



To start the Emulator Analysis tool:

Step 1: Load your program using the CCStudio IDE.

Step 2: Select Tools→Emulator Analysis for your device from the CCStudio 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.

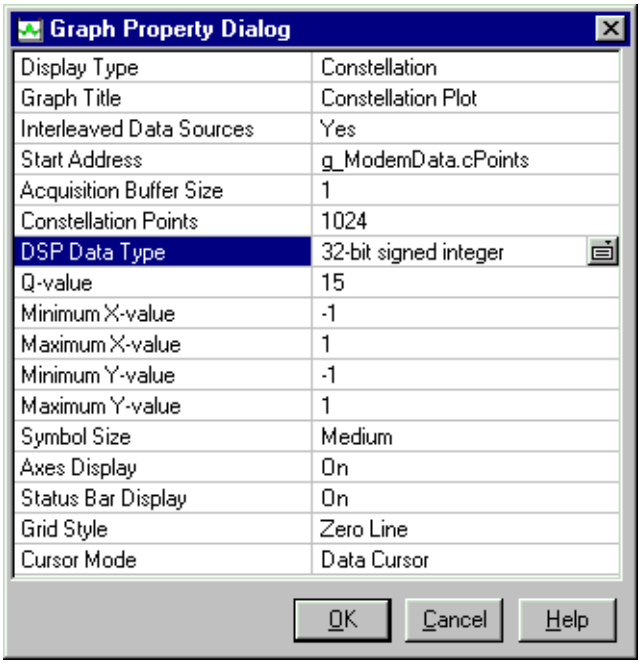
3.9 Displaying Graphs

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.

CCStudio 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.



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.

3.10 Symbol Browser

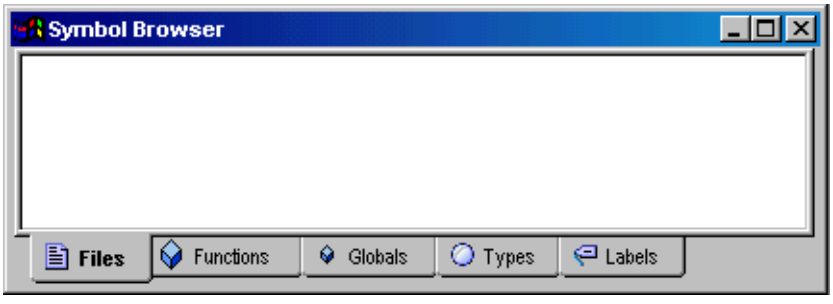
The Symbol Browser displays all of the associated files, functions, global variables, types, and labels of a loaded COFF output file (*.out). The Symbol Browser contains five tabbed windows:

- 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 3–6. 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.

3.11 General Extension Language (GEL)

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

You create your GEL functions using the GEL grammar, and then load them into CCStudio IDE. With GEL, you can access actual/simulated target memory locations and add options to CCStudio 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.

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

3.12 Command Window

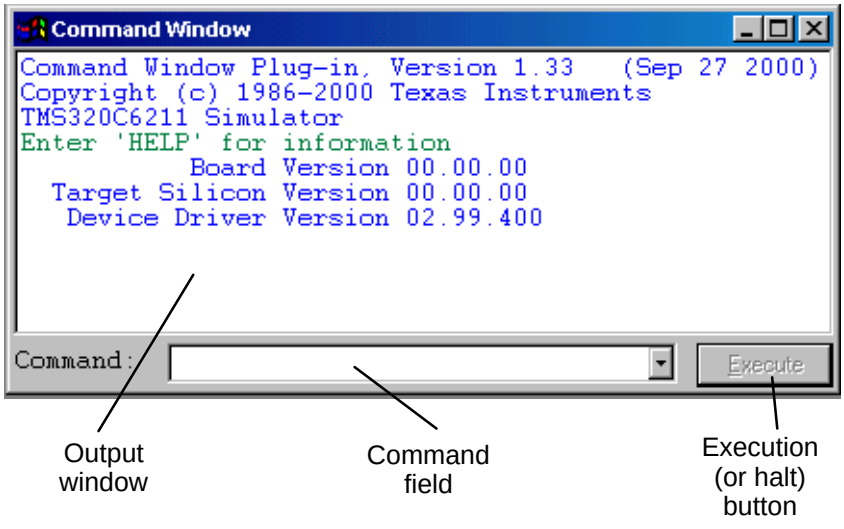
The Command Window enables you to specify commands to the CCStudio debugger using the TI Debugger command syntax.

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 CCStudio menu bar.

Figure 3–7. Command Window



For detailed information on the Command Window, see the Command Window topics provided in the online help: Help→Contents→Command Window.

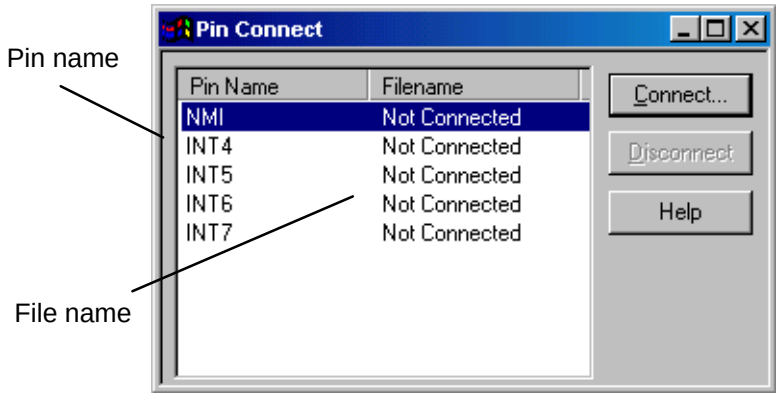
3.13 Pin Connect

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.



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.

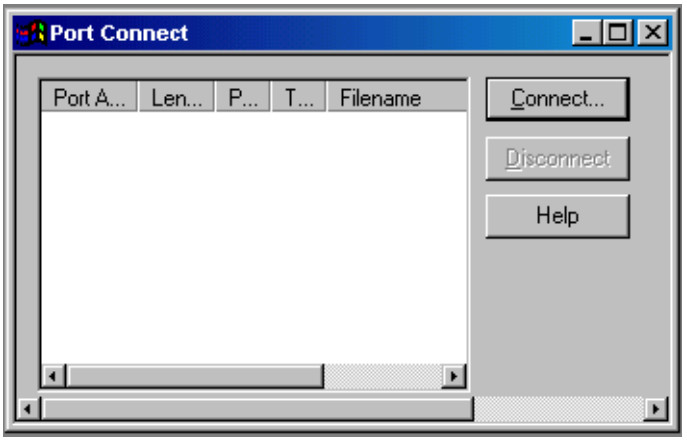
3.14 Port Connect

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:

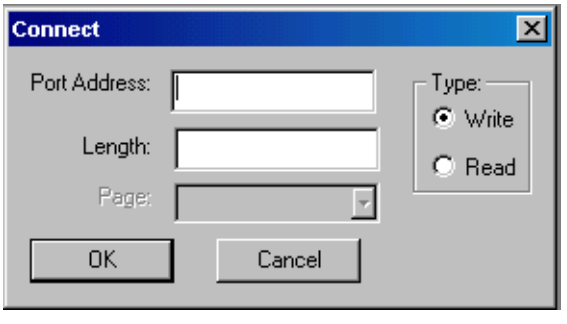
Step 1: From the Tools menu, select Port Connect.

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, CCStudio 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.

Chapter 4

Code Composer Studio for Solaris Optimization Tools

This chapter applies to all platforms using Code Composer Studio™ (CCStudio) 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 CCStudio 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 CCStudio profiling tools and shows you how to use them.

Topic **Page**

4.1 Profiler 4-2

4.1 Profiler

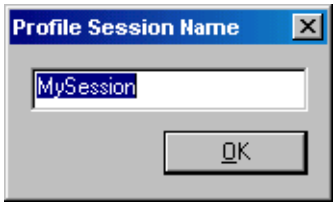
Profiling helps reduce the time it takes to identify and eliminate performance bottlenecks. The profiler analyzes program execution and shows you where your program is spending its time. For example, a profile analysis can report how many cycles a particular function takes to execute and how often it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

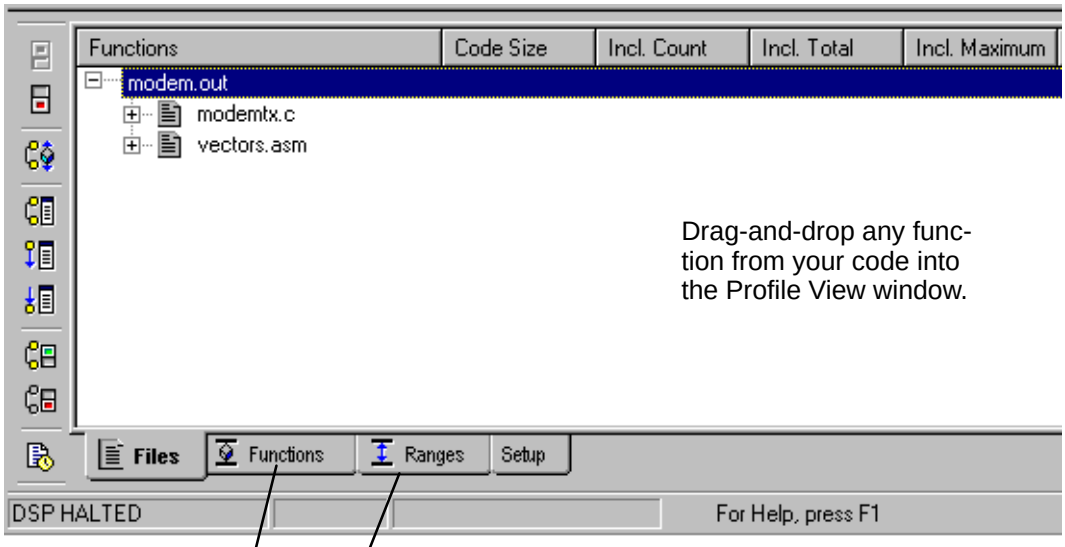
Step 1: Select Profiler→Start New Session.



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 CCStudio window.



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.

Chapter 5

Code Composer Studio for Solaris DSP/BIOS Kernel

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

DSP/BIOS kernel is a scalable kernel. DSP/BIOS kernel provides pre-emptive multi-threading and hardware abstraction.

DSP/BIOS kernel is 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, DSP/BIOS TextConf 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 take advantage of the multi-threading capabilities of DSP/BIOS kernel are implicitly instrumented.

DSP/BIOS kernel is integrated with CCStudio IDE and requires no runtime license fees.

Topic **Page**

5.1 DSP/BIOS Kernel Components 5-2

5.2 DSP/BIOS TextConf 5-3

5.3 DSP/BIOS Kernel 5-6

5.1 DSP/BIOS Kernel Components

DSP/BIOS kernel includes the following components:

- **DSP/BIOS Configuration Tool.** This easy-to-use visual editor 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. (Available on Windows only.)
- **DSP/BIOS TextConf.** You write and run JavaScript code that configures the DSP/BIOS kernel objects used by your program. You can use this tool to configure all the objects configurable with the DSP/BIOS Configuration Tool, but with more power and flexibility.
- **DSP/BIOS Kernel.** The DSP/BIOS kernel implements run-time services that the target application invokes through DSP/BIOS APIs.
- **DSP/BIOS Real-time Analysis Tools.** Several windows allow you to view the real-time behavior of an application that uses DSP/BIOS. For example, the Execution Graph shows a diagram of thread activity. Information captured by DSP/BIOS instrumentation can provide invaluable insights into the sequence and timing of events. These tools are most important later in the development cycle, when subtle problems may arise due to time-dependant interactions between program threads. (Available on Windows only.)
- **Chip Support Library.** The Chip Support Library provides macros and functions that simplify configuration and management of on-chip peripherals. See Chapter 6 for more information.

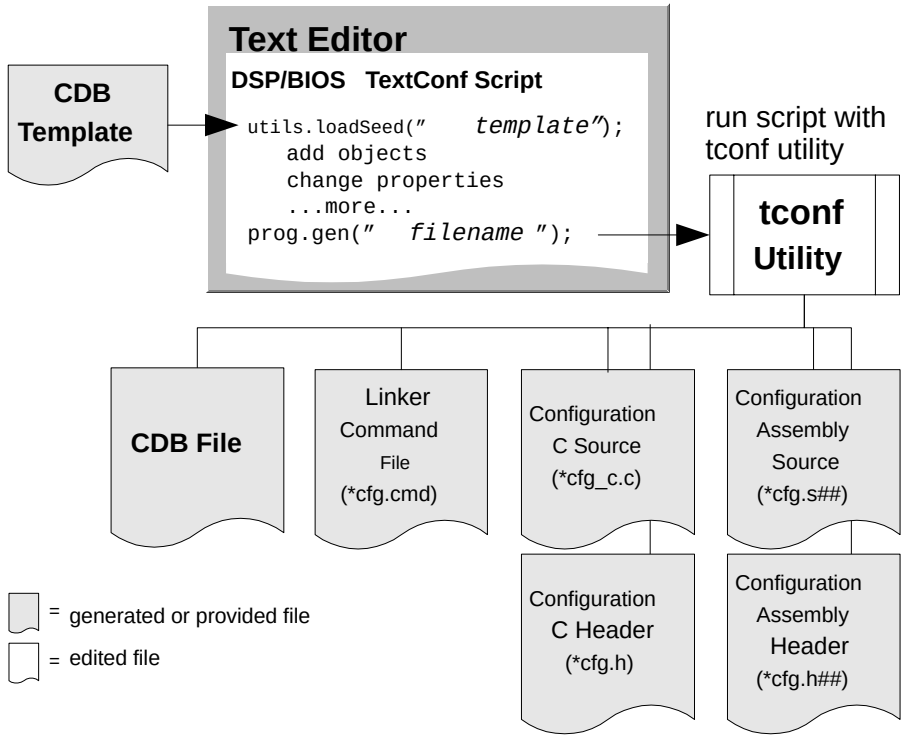
5.2 DSP/BIOS TextConf

DSP/BIOS TextConf is a JavaScript-based configuration language with associated utilities for running and generating scripts. It is especially useful for experienced users of DSP/BIOS and for applications that require powerful and flexible configuration solutions. (New users of DSP/BIOS may prefer to use the DSP/BIOS Configuration Tool available on Windows when learning about DSP/BIOS objects.)

DSP/BIOS TextConf allows you to write scripts that create and configure the DSP/BIOS objects used by your program. Your scripts can configure thread priorities, interrupt handlers, memory, Chip Support Library (CSL) settings, and more. Command-line utilities run and generate DSP/BIOS TextConf scripts. The scripts use the standard JavaScript language. You can edit and manipulate the scripts as you would source code.

As the following figure illustrates, typical DSP/BIOS TextConf scripts begin by loading a CDB template, contain statements that add objects and change properties, and then generate files to be used in building the application.

Figure 5–1. File Generation by DSP/BIOS TextConf



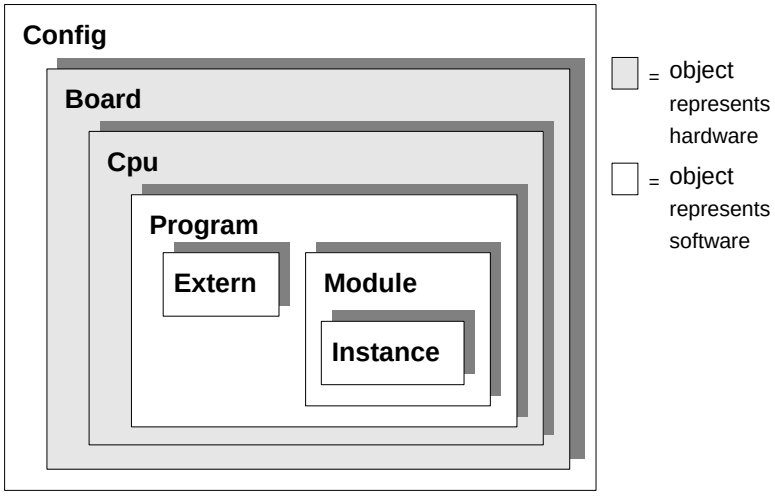
For example, the following statements are the TextConf script for a Hello World example:

```
utils.loadSeed("board_template.cdb");

var trace = LOG.create("trace");
trace.bufLen = 32;
LOG_system.bufLen = 128

prog.gen("myApp");
```

This script loads a board-specific CDB template file that defines the target type, memory objects, hardware interrupt objects, and more. The script then creates a LOG object called “trace,” and sets the buffer length of two LOG objects. Finally, it calls a method that generates a CDB file and source, header, and linker command files to be used when compiling and linking the application.

The JavaScript language used with DSP/BIOS TextConf adheres to the ECMA-262 standard. The object model implemented for DSP/BIOS TextConf is called the Target Content Object Model (TCOM). It includes Config, Board, CPU, Program, Module, and Instance objects. It provides configuration support for applications with multiple boards, CPUs, and programs.

Figure 5–2. Target Content Object Model (TCOM)



The JavaScript interpreter, the tconf utility, can be run in batch, interactive shell, or graphical mode.

DSP/BIOS TextConf statically declares and configures DSP/BIOS kernel objects during program development rather than during program execution. Such static objects exist for the duration of the program. DSP/BIOS kernel also allows many object types to be created and deleted dynamically during program execution. However, dynamically-created objects require additional code to support the dynamic operations. Statically declared objects minimize the memory footprint by excluding this additional code.

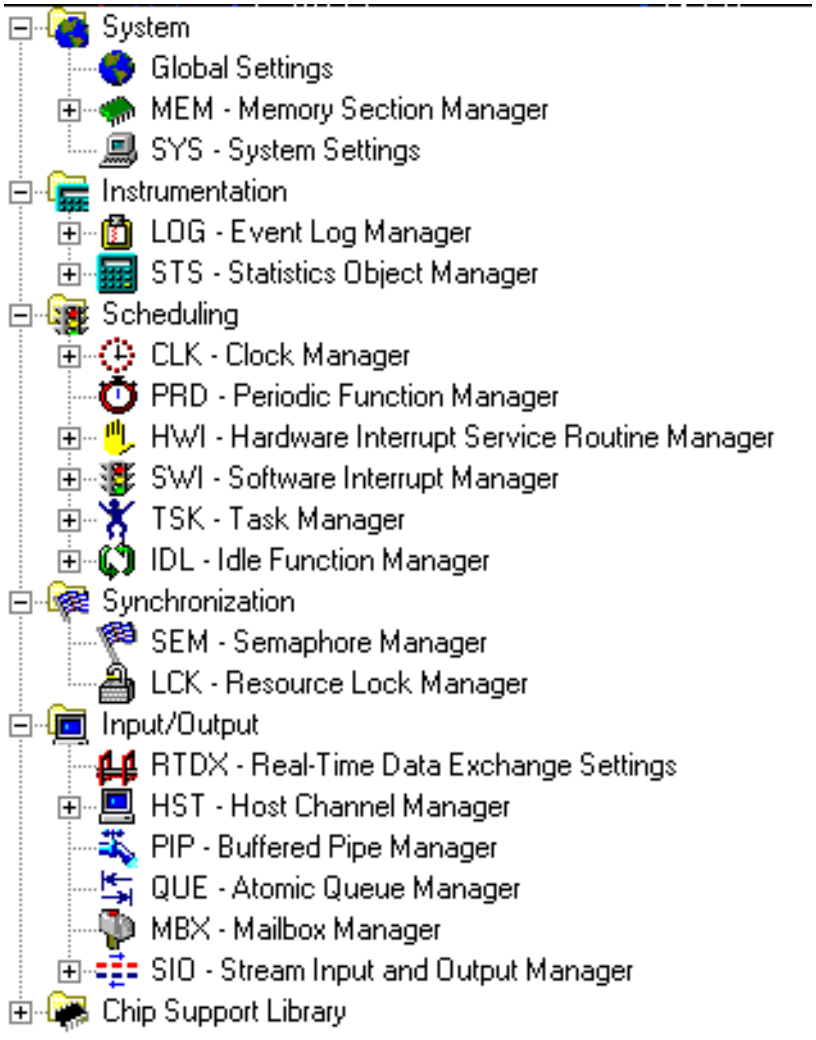
5.3 DSP/BIOS Kernel

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.

DSP/BIOS contains the following modules for which objects can be created. Additional modules provide atomic functions, target-specific functions, and trace control.

Figure 5–3. DSP/BIOS Modules



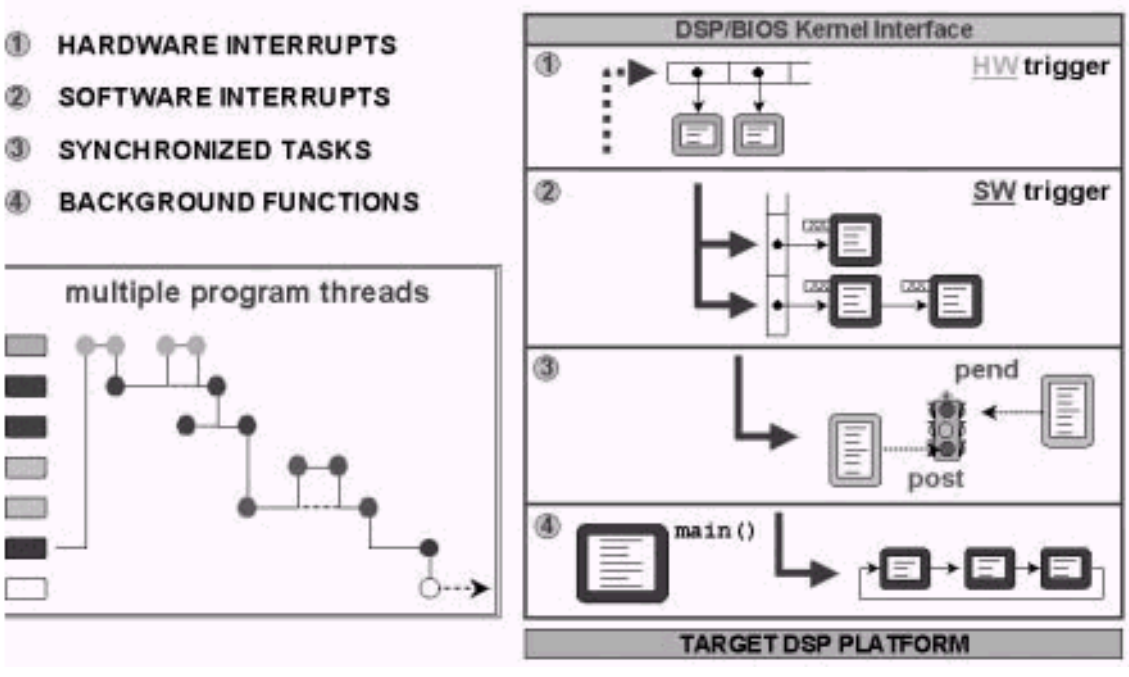
Developers can statically declare and configure many of these objects using DSP/BIOS TextConf. Developers may also declare and configure many of these objects dynamically within their programs.

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, DSP/BIOS TextConf 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 5–4). 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 5–4. 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 5–4). 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.

Chapter 6

Code Composer Studio for Solaris Chip Support Library Overview

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

Topic **Page**

6.1 Introduction to CSL 6-2

6.2 CSL Configuration Files Generated by DSP/BIOS TextConf 6-3

6.1 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.

DSP/BIOS TextConf (Chapter 5) provides an interface for configuration of the CSL. The CSL modules and instances allow some peripherals to be pre-initialized in the generated C files. The peripherals are pre-configured with the predefined objects.

Note:

If you use CSL without using DSP/BIOS TextConf or the DSP/BIOS Configuration Toold, CSL can conflict with DSP/BIOS functionality.

6.2 CSL Configuration Files Generated by DSP/BIOS TextConf

DSP/BIOS TextConf generates two C source files that contain CSL configuration statements. These files are the *project*cfg.h header file and the *project*cfg_c.c source file. In these filenames, project typically matches the name of the target executable filename.

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.

Chapter 7

Troubleshooting

This chapter provides usage hints, guidelines for troubleshooting, and answers to frequently asked questions.

Topic **Page**

7.1	Installation/Loading Code Composer Studio	7-2
7.2	File and Directory Names	7-6
7.3	General Debugging	7-6
7.4	Editor	7-7
7.5	General Extension Language – GEL	7-7
7.6	Graph Window	7-9
7.7	Miscellaneous	7-10

7.1 Installation/Loading Code Composer Studio

1) **Note: System Operation using XDS510WS emulator.**

   Turning off the XDS510WS device attached to the SCSI port while the workstation is running may cause a SCSI bus error and system crash.

2) **Note: Restarting Code Composer Studio.**

   In the rare event that CCStudio aborts or after using kill to terminate the process cc_app, the script “cc_cleanup” must be executed. Failure to run this script prevents CCStudio from starting properly in subsequent invocations.

3) **How can I quickly check that my workstation has all O/S patches required for CCStudio for Solaris?**

   Run the script os_patch.check.sh. This script is provided in the Sun_OS_patches directory of the product CD, and in the <install_dir>/ti/solaris/cc/bin directory of the product installation.

4) **Code Composer Studio does not start up, and no error message displays.**

   There may be two causes of this error:

   - Another cc_app process owned by the same user is running on the same system.
   - A previous invocation of CCStudio has been terminated abnormally and did not clean up semaphores.

   To alleviate the problem, run script “cc_cleanup,” then restart CCStudio.

5) **When I attempt to execute Code Composer Studio for the first time, I intermittently receive the following error messages:**

   - “Can’t Initialize Target DSP Trouble with JTAG controller
     Check your Cabling and your Multiprocessing Configuration”
   - “Can’t Initialize Target DSP
     I/O port – <address>”

   There are several troubleshooting areas to consider when encountering this error. These are listed below from most likely to least likely to have invoked this error message.

DSP Target I/O Settings:

- Your DSP target has been set an an invalid I/O address. Make sure that your DIP switch settings on your target card match the I/O address set when running the Code Composer Setup utility.
- A conflict may exist at the I/O address set for your target. Check that no other hardware on your workstation is using this I/O setting.
- Make sure that you did not set up multiple Simulator boards that use the same simulator core. Such combinations are not supported.

Code Composer Setup Configuration:

- Your Multiprocessing configuration has not been set up correctly. Please refer to the Code Composer Setup online help for details on correctly configuring your Code Composer Multiprocessing System.
- The Code Composer Setup utility may not reside in the same directory as your Code Composer Studio executable program. Make sure the setup program is in the same directory as the executable.

DSP Target Setup:

- Make sure that your DSP is not in a “Hold” or a “Reset” state and is correctly powered up.
- Target processor pin is active. The target processor must be “ready” for the debugger to execute. If there is a hardware problem on the ready line, if possible, put the processor into Microcomputer mode, reset the system, and try bringing up the debugger again. In Microcomputer mode, all memory accesses should ben on-chip and the ready signal should have no effect. Check your device user’s guide for details.
- The processor hold pin is active. Same issue as the target process pin being active.
- The JTAG signal may not be clear enough. In order to provide high-quality signals between the emulator and the target processor, check that the unbuffered distance between the emulator header and the processor is less than 6 inches. if this distance is in excess of 6 inches, the emulation signals should be buffered.

- The processor does not have a clock out. Processor must be receiving and generating the proper clocks. Check you clock in circuit and clock mode.
- The EMU0/1 pins must be high. The value of the EMU0/1 pins and reset can be used to turn off device pins and/or invoke device test modes. The user should have these pins pulled high through a resistor in his/her target system.

6) **When I attempt to run cc_setup or cc_app, I receive the following error message:**

"Error: Can't write to $DISPLAY:
Please set your $DISPLAY variable properly.
If you are running this script from a remote machine then make sure that the remote machine is allowed to open the display on the local machine."

There may be the following causes of this problem:

- The DISPLAY variable in the Unix shell might not be set. To check this, execute the following command:

  ```
  echo $DISPLAY
  ```

  If the variable is not set, the user running a C shell can set it as follows:

  ```
  setenv DISPLAY local_machine:0.0
  ```

  If running an sh-based shell, an equivalent command is:

  ```
  DISPLAY=local_machine:0.0
  export DISPLAY
  ```

- If you are trying to run cc_setup or cc_app on a remote machine, it might be that the remote workstation is not authorized to send an application window to the local machine. To alleviate the problem, execute the following command in the Unix shell on the local machine:

  ```
  xhost +remote_machine
  ```

- A rare but possible cause of the problem might be a previous crash of CCStudio, CC Setup, or another MainWin application. To alleviate the problem, execute command:

  ```
  cc_cleanup
  ```

7) **Some secondary pop–up windows drop behind the main CCStudio window when thy lose focus.**

By default, Sun Microsystems ships the Dtwm window manager with the option for secondary pop–up windows to stay on top of the main window disabled, even if they lose focus. Therefore, when the main window takes focus from a secondary window, the undocked secondary window drops behind the main window and is not easily accessible.

To allow for proper 2 level Z-ordering for pop-up windows, examine the .Xdefaults file in your home directory. If this file does not exist, create it. Place the following line in .Xdefaults, unless such line already exists.

Dtwm*secondariesOnTop: True

For the change to take effect, do one of the following:

- Log out from the workstation, and log in again
- Execute command “xrdb –merge .Xdefaults”, then right-click on the desktop and select Windows→Restart Workspace manager.

If you are running CCStudio for Solaris remotely with a local display on another Solaris workstation, the modification of .Xdefaults is required on the client (local) workstation and not on the server workstation.

7.2 File and Directory Names

Note: Files and directories whose names start with period (for example, .Code-Composer) are not displayed in dialogs such as File→Open. To open such files and directories, type their name in the file selection editbox.

7.3 General Debugging

1) **When I set a breakpoint at a valid C-line in my Edit window, I get an error message when I begin executing my program:**

   "Unable to move breakpoint to a valid line at source line: <filename> at line xxx. It has been disabled."

   It may be possible that:

   - You have not loaded the program onto the DSP. The CCStudio debugger needs you to load the program onto your DSP target ot get all the symbol information. This symbol information tells CCStudio the exact DSP address for each C source line.
   - No valid assembly line exists for the specified source line.

     NOTE: If you are having trouble setting breakpoints, the best way to figure out what is wrong is to turn the Mixed C/Assembly option on under the View menu. This display shows you all the C-lines as well as the associated DSP instructions. If you do not see any assembly lines in your file, no symbol information exists for this C-file.

2) **I cannot Step Out from the printf() function.**

   Since standard C functions (for example printf()) do not contain debugging information necessary to step out, the Step Out function does not work from within such functions. The execution just runs to the next breakpoint.

3) **A program was built without errors or warnings, but when I load it and try to Go Main, the execution never stops.**

   Make sure that you have added a linker command file to the CCStudio project. An example linker command file (lnk.cmd) is supplied in the following directory:

   <install_dir>/ti/solaris/c6000/cgtools/lib (for C6000 product)

   <install_dir>/ti/solaris/c5500/cgtools/lib (for C5000 product)

7.4 Editor

How do I shift a whole paragraph one Tab position in the editor?

Use the mouse to select the paragraph of interest. Then, use the Tab (or shift-Tab) to shift the entire paragraph one tab position.

7.5 General Extension Language – GEL

1) **Why are quotation marks needed around some symbols in some places and not others? For example, WatchAdd requires quotes and BreakPtAdd does not?**

   Quotes are placed around an expression or string that is NOT to be parsed or evaluated before it is executed.

   Whenever you type in an expression, it is parsed and executed. The same is true if you call a GEL function (including built-in functions). If you pass an expression as an argument to a GEL function, it will be evaluated and the final result will be passed to the GEL function.

   For example, in the following call to the built-in function:

   GEL_BreakPtAdd(StartAddress + 0x100)

   the expression “StartAddress + 0x100” is evaluated and then the result is passed to the built-in function. For the built-in function GEL_WatchAdd(), we do not want to pass an expression as an argument. We want to pass a STRING that is added to the Watch window. It is the Watch window that is responsible for evaluating the expression contained in the string.

2) **Can I execute a GEL function each time I start Code Composer Studio?**

   Yes. You can automatically load a GEL function upon startup if you specify a Startup GEL File name in the Board Properties for your target configuration in the Code Composer Studio Setup.

   Note that this action is only LOADING the GEL file into CCStudio’s memory so you can access the GEL functions within the file. However, if this file contains a GEL function named “StartUp()”, CCStudio will execute this function. Therefore, you can place your initialization tasks within this function.

3) **How can I execute a GEL function every time the code hits a breakpoint?**

   A good way to execute a GEL function at a breakpoint is to set a conditional breakpoint.

   Enter the function you want to call as the conditional expression. If you just want to call the GEL function and then continue to execute, have the GEL function return FALSE (ie, 0); otherwise, return TRUE (ie, 1).

4) **How do I automatically load a GEL file and a Workspace every time I invoke Code Composer Studio?**

   Within the CCStudio environment, load your GEL files and set up your debugging environment. Then, save the setup as a workspace and place this workspace name (including the .wks extension) on the CCStudio executable command line.

7.6 Graph Window

1) **Can I prevent graphs from updating at least when I am stepping through code?**

   Yes. What you need to do is connect the graph to a Probe Point. If a graph is not connected to a probe point (which is the case when you first open a graph window), it is updated each time you single-step or run to a breakpoint. However, if you connect it to a Probe Point, it only updates when the execution of the code reaches the probe point.

2) **I would like to trace the values of a variable in the Graph window. Is this feature supported in Code Composer Studio?**

   Yes. The Graph window is flexible enough to allow you to trace values of a variable. These are the steps you should follow:

   - Open a Graph window with the following parameters:

Starting Address	If your variable of interest is “ErrorPower”, then enter “&ErrorPower” for this location. The most common mistake here is when you do not enter “&” before the variable. Note that any valid ‘C’ expression can be entered in this field and the result should give you the address of the variable that you are interested in.
Buffer Size	Enter ‘1.’
Display Size	Enter the number of samples you want history for.
Left Shift Display	Select this option

   - Connect the Graph window to a probe point. When you use the left shift display option, it is recommended that you attach the graph to a probe point so the Graph window does not update unless you want it to. The graphical display now graphs the values of the variable.

7.7 Miscellaneous

1) **Note:** To redock a dockable window after it has been moved onto the desktop, right-click on it and check Allow Docking. After that, place the mouse pointer on the thin gray area just below the title bar, press the left mouse button, and drag the window into the parent CCStudio window.

2) **Note:** In the TI Command Window plug-in, the command “cd” with no parameters reports the current directory. It does not change to the home directory, unlike the UNIX “cd.”

3) **Note:** In the TI Command Window plug-in, register names must be entered in all uppercase letters.

4) **I’m using Solaris 2.6 with a non-POSIX locale and Code Composer Studio does not start.**

   Set your locale to POSIX (”C”) at the CDE login screen. There is a bug in locales with Solaris 2.6.

5) **Code Composer Studio does not display colors correctly.**

   If you are using a web browser or any other application that sets many colors in the palette, then close that application and run CCStudio before you start up the web browser or other application. If this does not solve the problem, make sure that your default X visual is set to “PseudoColor.” You can set your default X visual to “PseudoColor” by editing the file:

   ```
   /usr/openwin/lib/X11.sdm/Xservers
   ```

   Look for a line similar to:

   ```
   :0 local /usr/openwin/bin/X –dev /dev/fb defdepth 24
   defclass TrueColor
   ```

   The line in your file may not be exactly the same as above. Change the defclass parameter to PseudoColor or the defdepth to 8. That is, either:

   ```
   :0 local /usr/openwin/bin/X –dev/fb defdepth 24 def-
   class PseudoColor
   ```

   or

   ```
   :0 local /usr/openwin/bin/X –dev/fb defdepth 8 defclass
   TrueColor
   ```

6) **I know that my source files and makefile are correct, but Code Generation Tools produce error message(s) when I try to build a project.**

   If you did not use the setpath.csh or setpath.sh scripts from <install_dir>/ti/solaris/cc/bin to set environmental variables for Code Generation Tools, make sure that they are set exactly as shown in those scripts. For instance, the TI_DIR variable must be without a trailing slash, i.e. <install_dir>/ti/solaris but not <install_dir>/ti/solaris/.

7) **I cannot remove the old installation tree of CCStudio for Solaris with the Unix "rm –rf" command, even though I own all directories within this tree.**

   The CCStudio for Solaris directories are installed as read-only. You have to make these directories writeable for the owner before deleting them with the "rm –rf" command. Execute command:

   ```
   chmod –R u+w <install_dir>/ti
   ```

   and then

   ```
   rm –rf <install_dir>/ti
   ```

Index

A

absolute lister 2-4
accessing documentation 1-24
API
  DSP/BIOS 5-2, 5-6
  execution threads 5-7
archiver, described 2-4
assembler
  described 2-4
  overview 2-5
assembly language tools 2-4
  absolute lister 2-4
  archiver 2-4
  assembler 2-4
  cross–reference lister 2-4
  hex–conversion utility 2-4
  linker 2-4
assembly optimizer, described 2-9

B

bookmarks, code window 3-5
breakpoints
  hardware 3-8
  introduction 3-6
  software 3-7
  vs. probe points 3-13

C

C/C++ code development tools 2-9
C/C++ compiler, described 2-9
Chip Support Library 6-2
  DSP/BIOS TextConf 6-3
  header file 6-3
  source file 6-4
code, review using the editor 3-4
Code Composer Studio Tutorial 1-23
code development tools
  assembly optimizer 2-9
  C++ name demangling utility 2-9
  C/C++ 2-9
  library–build utility 2-9
  run–time–support libraries 2-9
code generation tools 2-2
code window
  bookmarks 3-5
  keyboard shortcuts 3-5
  keywords 3-4
  selection margin 3-4
Command Window 3-24
compiler, overview 2-3 to 2-6
Configuration, create a 1-11
configuration tool, DSP/BIOS 5-2
Create a System Configuration 1-11
cross-reference utility 2-4
CSL
  DSP/BIOS TextConf 6-3
  header file 6-3
  Introduction to 6-2
  source file 6-4
CSL (chip support library), benefits of 6-2
CSL Benefits 6-2

D

debug tools
  Command Window 3-24
  emulator analysis 3-19
  graphs 3-20
  overview 3-2
  Pin Connect 3-25
  Port Connect 3-26
  probe points 3-13

simulator analysis 3-17
symbol browser 3-22
Watch window 3-9
documentation, accessing 1-24
DSP/BIOS
API 5-2, 5-6
configuration tool 5-2
execution threads 5-7
kernel 5-2, 5-6
TextConf 5-3
using 5-1

E

editor, review source code 3-4
emulator analysis 3-19
EPROM programmer 2-4
execution threads 5-7
external editor, source code 3-5

G

GEL, general extension language 3-23
graphs, displaying 3-20

H

hardware breakpoints 3-8
header file, CSL 6-3
hex conversion utility, described 2-4

I

Import Configuration dialog box 1-11, 1-12
interlist utility, described 2-9

K

kernel, DSP/BIOS 5-2, 5-6
keyboard shortcuts, code window 3-5
keywords, code window 3-4

L

library-build utility, described 2-9
linker
described 2-4
overview 2-6

O

optimization tools, Profiler 4-2
optimizer, described 2-9
overview
assembler 2-5
debug tools 3-2
linker 2-6

P

Pin Connect 3-25
Port Connect 3-26
probe points 3-13
vs. breakpoints 3-13
Profile session 4-2
Profiler 4-2
program, run 3-3
project run 3-3

R

real–time component, DSP/BIOS 5-1
run program 3-3
run project 3-3
runtime-support, library, described 2-9

S

selection margin, code window 3-4
session, for the Profiler 4-2
shell program, described 2-9
simulator analysis 3-17
user options 3-17
software breakpoints 3-7
source code
external editor 3-5
review 3-4
source file, CSL 6-4

symbol browser 3-22
System Configuration, Create a 1-11

T

TextConf, DSP/BIOS 5-3
threads, execution 5-7
tools
  assembly language 2-4
  code generation 2-2
  compiler 2-3
tutorial 1-23

U

using, DSP/BIOS 5-1

W

Watch window 3-9
  display 3-9
  toolbar 3-9