Using Example Projects, Code and Scripts to Jump-Start Customers With Code Composer Studio 2.0

Steve White, Senior Applications *Code Composer Studio, Applications Engineering*

ABSTRACT

This application note describes the new and improved features of Code Composer Studio™ (CCS) 2.0 and provides instructions on how to use the new features. This is done by utilizing existing examples and sample code that are shipped with Code Composer Studio 2.0. This note attempts to correlate the new features with the Code Development cycle.

Contents

List of Figures

1 Introduction

This application note uses existing code and sample files that ship with Code Composer Studio 2.0 to walk through the code development cycle. It also highlights the new features that are part of this release and utilizes them while explaining the development cycle.

The purpose of this application note is to provide an introduction to Code Composer Studio 2.0, and allow developers to acquaint themselves with the latest features offered by the IDE.

For the purpose of this application note, the Code Composer Studio 2.0 simulator is used, configured as a TMS320C64x™, (C64x™) using fast sim and little-endian configuration. For simplicity, the HELLO2 tutorial that is included with Code Composer Studio is used. You should also bear in mind that the following steps and principles can be applied to actual DSP hardware targets and most ISAs (Instruction Set Architectures).

TIP – The Code Composer Studio simulator does not support multiple processors.

TIP – Code Composer Studio does not affect, add or modify any environment variables nor is there a need for you to make any additions or changes to the autoexec.bat file (depending on whether you are using Windows 98, Windows 2000 or Windows NT). This information is stored in the registry. If you need to build from a command line or set any environment variables, please run the batch file DosRun.bat (generated and located in the c:\ti directory during default installation).

2 Code Composer Studio and the Code Development Cycle

2.1 The Code Phase

Code Composer Studio is an integrated development environment that is tuned for DSP development that can assist you in many ways during the entire development cycle. Code Composer Studio enables you to decrease time to market and it shortens your learning curve.

In this section, the existing features of Code Composer Studio are covered as well as the new functionality included with the product. The features covered include the Editor, Project Manager, Visual Linker, PBC (profile based compilation), GEL (General Extension Language), Source Control and the extensive online help as they relate to writing your application.

2.2 The Debug Phase

This section walks you through a typical debugging session with example code that is provided with the Code Composer Studio software. The features highlighted are the Watch Window, GEL (General Extension Language) and Breakpoints. Symbol Browser functionality of the software is covered.

TMS320C64x and C64x are trademarks of Texas Instruments.

2.3 The Analyze Phase

During the analysis phase of the development cycle, you will be concerned with scheduling and Real Time Analysis in order to ensure your code is correct. This section covers DSP/BIOS, which is a scalable, real-time kernel designed to work with applications requiring real-time scheduling, synchronization or instrumentation. Details are provided on the graphing capability of the software as well as Advanced Event Triggering and Advanced Breakpoints.

2.4 The Optimization Phase

This section details how you can use the new Code Composer Studio Profiler and PBC (Profile Based Compilation) to benchmark your code for optimal performance and size. Todays' DSP applications need to keep in mind the memory limitations and speed of new and existing hardware. The optimize phase of the cycle covers how to trim code to optimal size and ensure it runs efficiently on the hardware it is written for.

2.5 New Features of Code Composer Studio 2.0

This section goes into some detail about the functionality that is new to Code Composer Studio 2.0. It also describes new features that have been included in moving from Code Composer Studio 1.2 to Code Composer Studio 2.0. Information is provided on features such as Profile Based Compilation (PBC), Advanced Event Triggering, DSP/BIOS, Symbol Browser, Source Control, and External Editor. The goal is to tie all these new features and existing functionality into use of the Code Composer Studio environment while working through a simple development project. At the end of this application note is a glossary of features that are new to Code Composer Studio 2.0 and features that are new in going from Code Composer Studio 1.2 to Code Composer Studio 2.0.

3 Configuring and Launching Code Composer Studio

(*This section is recommended reading if you are new to Code Composer Studio.)

After installing the Code Composer Studio software, you will see that there are two Code Composer Studio icons on your desktop. One of these icons is to configure the software and the other is to launch the application.

Please double-click on the icon titled Setup CCS 2.0 – this opens up the Code Composer Studio Setup dialog box along with the Import Configuration dialog box. The two dialog boxes shown in Figure 1 appear and you have a choice as to what configuration you wish to choose. In the Import Configuration dialog, there is a list of configurations that are supplied with Code Composer Studio.

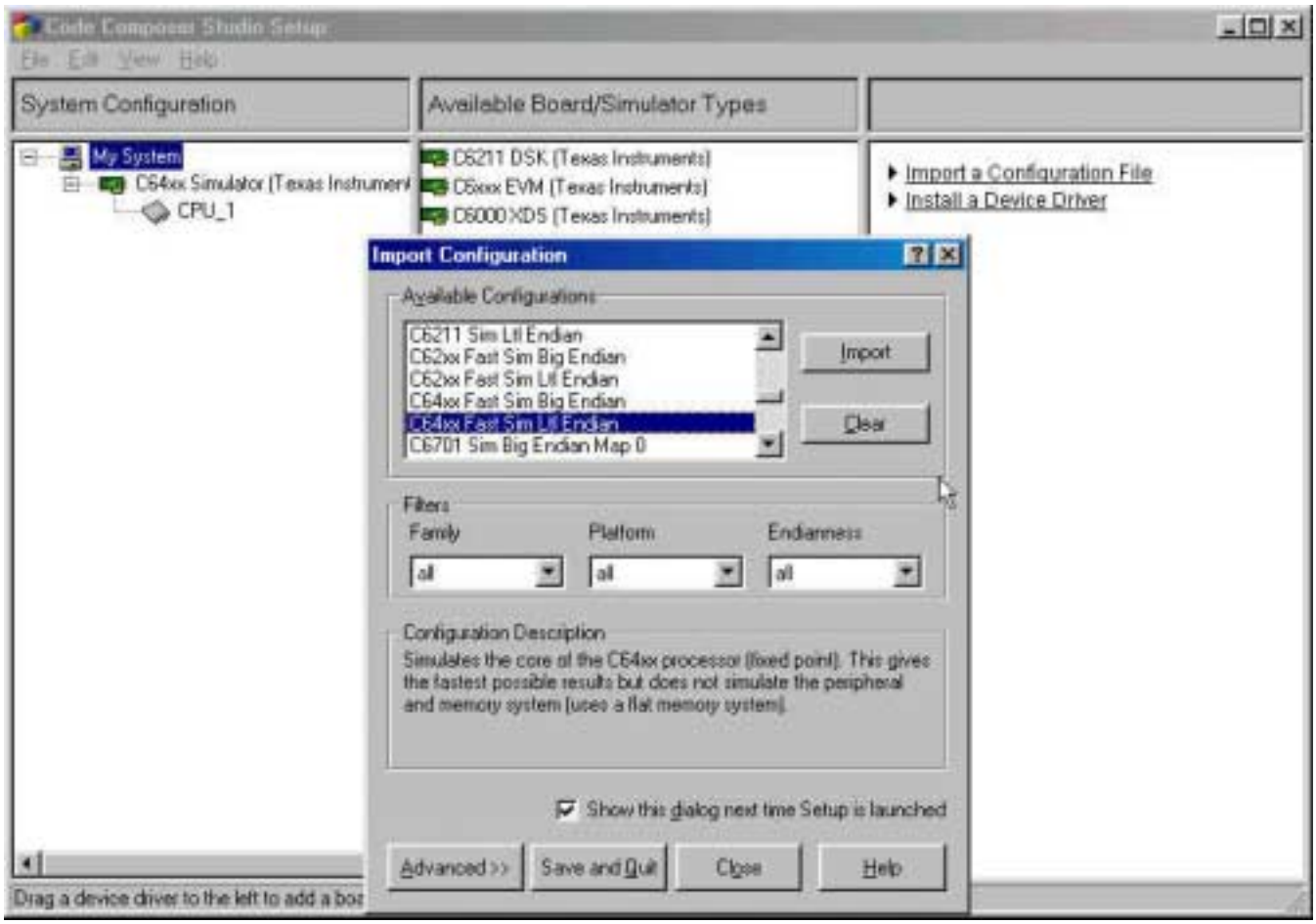

Figure 1. Code Composer Studio Setup Dialog and Import Configuration Dialog

3.1 Choosing Your Code Composer Studio Configuration

Once the dialog boxes shown in Figure 1 appear, you are ready to set up the software for use with your particular hardware. As mentioned in section 1, the examples this application note details are demonstrated on a C64x simulator. However, if you choose to not use a C64x simulator, you need to configure Code Composer Studio to use your hardware. Failure to configure the software correctly prohibits any communication between the host and target DSP.

Code Composer Studio Setup comes bundled with an assortment of configuration files that are designed to work with the most common system configurations. Simply go through the list and when you see a supplied file that matches your system configuration, highlight that file and click on the Import button in the Import Configuration dialog box. This action loads a configuration file into the Code Composer Studio setup utility and prepares to start a Code Composer Studio session.

There are three pull-down menus in the Filters section of the Import Configuration dialog as follows:

Family – Allows you to choose a particular family of DSPs, such as C6000, C64x or all included with this installation of Code Composer Studio.

Platform – Allows you choose the proper hardware platform that you are working with on a particular session such as DSK, EVM, Simulator, All, etc.

Endianness – Allows you to choose the type of endianness that you want to build your code with. The choices are: big, little or all.

When you highlight any configuration file in the Import Configuration dialog, the detailed information on that configuration file populates the Configuration Description field. This field is directly below the Filters section. This information tells you what DSP target the configuration file supports and other information. In some cases the description also informs you of the peripheral devices the file does and does not support, such as McBSP and DMA. The Advanced>> button offers some other functionality such as the capability to import configuration files from other directories and the ability to deal with duplicate filenames.

3.2 Code Composer Studio Setup

There are three panes on this dialog box (see Figure 1). Once you have imported the desired configuration file, it should appear in the leftmost pane of the Setup dialog under the System Configuration heading. The center pane of the Setup dialog provides you with a list of available board and simulator types. If you click on any of the available board or simulator types in the center pane of this dialog, the rightmost pane of the dialog is populated with details of the particular board/simulator type you have highlighted. The details presented are the location of the driver, revision of the driver, description, and boards supported by the driver. This information is very useful to know for not only you but also for the Technical Support organization if you need assistance.

By right-clicking on any board or simulator type in either the left-most or center panes, Code Composer Studio will present you with more options such as: remove board, rename board, properties, install, uninstall, and add to system.

When a board is highlighted, the Properties... option presents the Board Properties dialog box (see Figure 2).

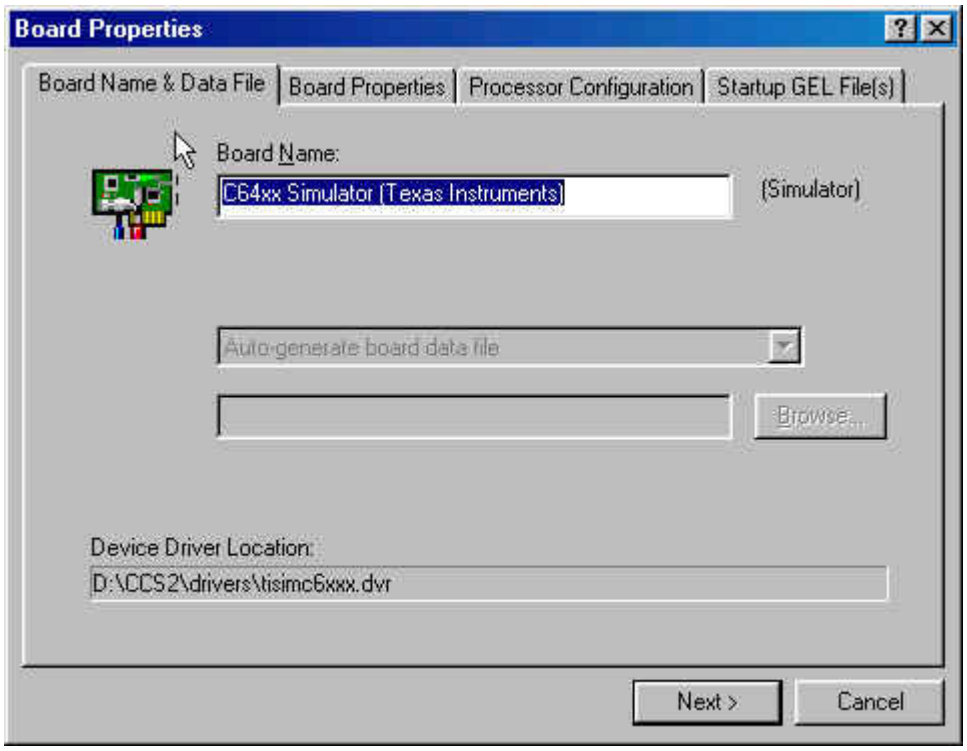

Figure 2. Board Properties Dialog Box

The first tab, Board Name & Data File, allows you to see what type of board is configured and the device driver location. The Board Properties tab informs you what configuration file and endianness the software is using. The Processor Configuration tab allows you to select a single processor, add multiple processors, or remove one or more processors from the configuration. The Startup Gel File(s) tab enables the running of a GEL (General Extension Language) file to initialize and configure the target DSP. This is required because sometimes, depending on the target DSP, certain registers and memory **must** be configured in order for the target DSP to receive and run code properly, and for proper operation of the DSP as well as debugging your code.

For more detailed information on any part of the setup procedure or component thereof, please consult the extensive online documentation. After going through the setup procedure, you are prompted to Save the configuration. If Yes is chosen, you are prompted to Start Code Composer Studio on exit. Answering Yes opens up the Code Composer Studio application while a No response simply saves your setup information without opening the application. Once you have chosen your configuration and saved it, this setup information is stored in the registry and in the ccBrd.dat file. This file is binary and is non-editable.

4 Before Beginning Your First Code Composer Studio Session

Now that you have configured this copy of Code Composer Studio for the DSP target you are planning to use, you are ready to set up the environment for the first coding session. Code Composer Studio allows for much flexibility as far as permitting you to use an external editor or source control, or to set hotkeys and shortcuts. In the following sections, some of the issues you may encounter in setting up the Code Composer Studio interface are detailed. Before the use of an external editor is discussed, it would be prudent to present some information on the Code Composer Studio built-in editor. As described below, there are some features that are available with the built-in editor that are not available when using a third party editor, such as advanced editor properties. However, the built-in editor does have some limitations regarding its use. The maximum number of characters per line is 3500 and the maximum number of lines per file is 2,147, 483, 648. The built-in editor also offers the capability of using bookmarks, which are an easy way to quickly locate a particular line of code in your source file. These bookmarks are used for location and/or maintainability purposes in a source file and they can be set at any line of any source file.

The built-in editor offers the following features that may not be available when using an external editor. The Code Composer Studio editor has had the file limitations removed in that you can work with unlimited file size and unlimited line length. The Selection Margin provides range markers for breakpoints, Probe Points, and Profile Points. You can do automatic code completion, view structure member listings, and have access to tool tip variable watching.

4.1 Choosing an External Editor

One of the new features of Code Composer Studio is the ability to integrate and use an external third party editor, such as Codewright, UltraEdit or VI. You may prefer to use an external editor that you are familiar with or as dictated by Company policy. For the purpose of this exercise, Codewright 5.1a is used. Once this external editor is enabled, the external editor launches whenever a new blank document is created or an existing document is opened.

Please bear in mind that this external editor can only be used to edit files. In order to debug the files, the Code Composer Studio integrated editor must be used. The Code Composer Studio editor is integrated with the Code Generation Tools and debugging tools. Therefore, certain operations such as set/clear breakpoints, set/clear Probe Points, single step, automatic locate of syntax errors can only be performed by the Code Composer Studio integrated editor. When it is time to debug, you must remember to disable the external editor in order to gain access to the debugging functions.

4.2 Enabling and Disabling the External Editor

In order to enable an external editor, you need to ensure that Code Composer Studio is open and then go to the Option menu. Once in the Option menu, at the bottom of the list is the Customize selection. Click on this selection and when the dialog opens, choose the Editor Properties tab (see Figure 3). Once you are on this tab, in the top right of this dialog there are options for External Editor Support. You need to ensure that the Use External Editor checkbox is checked and that the path to the editor is correct. If it is not known, clicking on the Browse button allows you to path out to the location of the external editor directory. After configuring your external editor, go to the Edit menu and down to the Enable External Editor option. When the external editor is enabled, a checkmark will be in front of the Enable External Editor command. Conversely, simply go to the Edit menu and uncheck the Enable External Editor option to disable external editor support.

Once you have set up Code Composer Studio to use an external editor in the Customize dialog box, just click on the Apply button and then the OK button for the changes to take effect.

See Figure 3 for a look at the Customize dialog box. For more detailed information on External Editor Support, please refer to the Code Composer Studio online help.

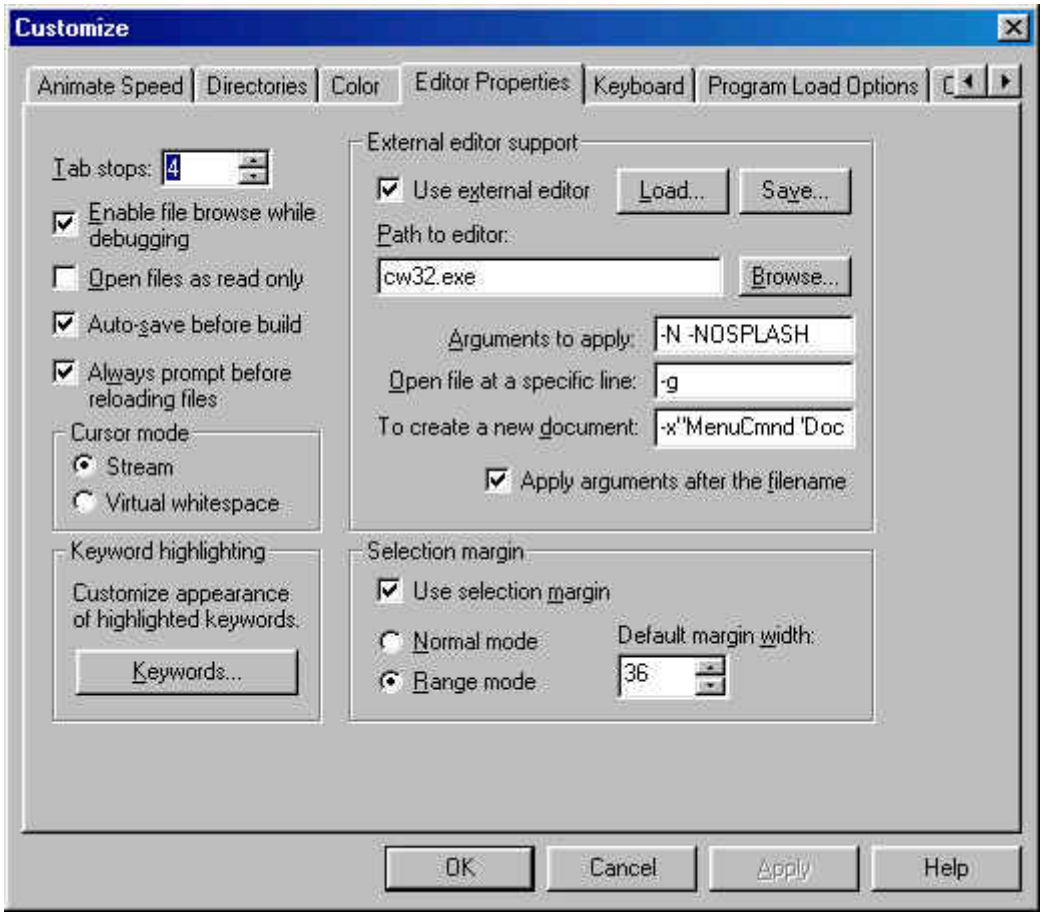

Figure 3. Customize Dialog Box With Editor Properties Tab

4.3 Other Tabs on the Customize Dialog Box

Animate Speed Tab – The first tab on the Customize dialog box is the Animate Speed tab. When you have loaded your executable code after compiling all your source files, and you run it continuously on the target DSP, this is called *animating*. Using this tab, you can set the minimum time between breakpoints that you have set in your code. When you are animating your code, execution does not resume until the minimum time has lapsed since the last breakpoint. This tab offers you the flexibility to set this minimum time or animate speed from one (1) to nine (9) seconds.

Directories Tab – This tab allows you to specify the path that the debugger uses to locate the source files that are included in your project. The directories list displays the defined search path and the debugger searches the directories listed in order from top to bottom. If two files have the same name but are in different directories, the file in the directory that is highest in the list has priority over the file in the lower directory. This dialog allows you to add a new directory, delete a directory already in the list or to move a directory in the list either up or down in priority.

Color Tab – This tab enables you to change the default colors of different screen elements of the Code Composer Studio environment. Once a color is changed and applied, this information is stored in the Code composer Studio workspace (covered later). You can change the colors of various texts, backgrounds, graph and axis backgrounds, grids, sets of keywords and function names.

Editor Properties – This tab not only allows you to choose and utilize an external editor as discussed in section 4.2, but enables you to set tab stops, browse files while debugging, open files as read-only, auto-save before build and prompt before reloading. You can also select the Cursor mode. In Stream mode your editor acts similarly to the way Microsoft Word selects and inserts text. In Virtual whitespace mode you can use the mouse or arrows keys to move the cursor to any place within the document window. The editor automatically fills in any whitespace between the cursor and the end of line with spaces. You can also set keyword color highlighting to suit your own development standards or according to target processor and file type, including GEL files. To do this, you can create your own custom keyword files or use one of the four provided keyword files. You can also choose the functionality of your selection margin, which by default, is displayed on the left side of the Code Composer Studio editor and on disassembly windows. There are colored icons in the selection margin that tell you where you have a breakpoint (red dot) or a Probe Point (blue dot) set at a particular location. A yellow arrow indicates the location of the program counter. This feature also allows you to display line numbers and marker points associated with a particular line of code. You can also change the width of the selection margin by entering a number between 20 and 999 as well as the mode of operation, to Normal mode or Range mode. For more detailed information on this dialog tab or any features listed, please consult the Code Composer Studio online help.

TIP – This functionality not available with an external editor.

Keyboard Properties Tab – There are a number of default keyboard shortcuts that come with Code Composer Studio that enable you to save time in performing repetitive or multiple keyboard/mouse commands by using a single key or combination of keys. Among the default shortcuts are the ability to: move insertion point, scroll and select text, delete, insert or copy text, window management, mark text, various debugging shortcuts and so on. You can also assign custom keyboard shortcuts or change the default shortcuts for any editing or debug commands that can be invoked from any document window. There is more information on using this functionality in the Code Composer Studio online help.

TIP – Please take note that these features are only available in document windows.

Program Load Options Tab – This tab allows you to select certain actions that are to automatically happen when programs or symbols are loaded onto the DSP or simulated DSP. The options available here are: Perform verification after Program Load, which enables Code Composer Studio to verify the program was loaded onto the target correctly. It does this by reading back selected memory from the target. The next option is Load Program After Build, which automatically loads your executable right after it has been built or modified. This guarantees that you have loaded the latest symbol information onto your target. Finally, the last option is Add Symbols With Offsets, which enables you to specify a code offset and a data offset when a new symbol file is loaded using the Add Symbols command. The symbol file specifies the code and data addresses where symbols are to be loaded.

Control Window Display Tab – The topmost section, Title Bar Displays, allows you flexibility in the information that is displayed in the title bar of the Code Composer Studio control window. In some cases, you may be working on several projects simultaneously. Some of these projects may have different board names, processor names or types and these may be specific to a product that is to be released. You may find it easier to administer and work with projects that have additional details in the title. The next section of this tab enables Code Composer Studio to display the full pathname to the source files. You can also select whether to close all windows of the project when the Project close option is selected or to close projects when exiting the Control Window.

File Access Tab – This tab allows you to select the maximum number of recent files to display in the list in the Code Composer Studio File and Project menus. Recent source files, Programs and Workspaces can be selected from the File menu. You can select from 1–10 files to display and the default is 4. You can also select between 1–10 (maximum) project files to display in the Project menu. Code Composer Studio allows you to Reset All File Directories when a project is opened by simply ensuring that box is checked.

Shared Memory Configuration Tab – The operations on this tab allow you to halt all processors with executable code on a shared block of memory to ensure that no breakpoints are missed and so that invalid code is not executed. The second option allows you to override the default action when stepping over a breakpoint that is set in a shared memory block. When stepping over a software breakpoint, the breakpoint has to be cleared first, then the processor steps, then the breakpoint is reset. Any other processors executing code in that shared memory block could miss the breakpoint. By default, when specified as a shared memory attribute, all processors that have executable code on that memory block are halted until the code has stepped over the breakpoint.

Advanced Editor Features Tab – This tab allows you to enable or disable certain features of the integrated editor. The Code Composer Studio editor prompts you with a tooltip as it attempts to recognize the word you are typing. The editor also tries to complete any words you are typing: simply type a few letters and hit the Tab key to be presented with a list of possible selections. The integrated editor also attempts to select the correct case for any keyword you are typing if this feature is enabled. You can enable Code Composer Studio to display a list of members available for an object by typing a period that separates the object name and the member name.

TIP – This functionality is not available when using an external editor.

5 Code Composer Studio and Source Control

Code Composer Studio Integrated Development Environment also allows you to implement any source code control software that utilizes the Microsoft SCC interface. For managing large and complex development projects, source code control is necessary. With projects such as these, often there are many engineers working on the project and these engineers can be at different geographic locations. Source Code Control tools provide a mechanism to keep track of all changes that are made to individual source files and to prevent files from being accessed by more than one engineer at a time.

Once you have chosen a source code control tool and configured Code Composer Studio to use it, those tools are used whenever any source code control commands are issued. This tool can also be changed at any time you wish. A Code Composer Studio project that is not under Source Control can be placed under it at any time. When a Code Composer Studio project has been associated with a source control tool, whenever the project is opened, it is automatically connected to the correct source control tool.

Whenever a project is opened that has not been placed under source control, you will be prompted to add the project to a new or existing source control project. You also have the option to leave the project without source code control.

5.1 Setting up Source Control

Open up the Code Composer Studio interface and go to the Project menu. From the Project menu, choose the Source Control option. Then go to the Select Provider option and choose the source control tool of choice that is available on your PC. Once you have selected your source control tool, in order to open or start the tool – go to the Project menu, Source Control option and select Launch Client Tool. A Source Control tool was not selected in this case, therefore, the dialog in Figure 4 is reporting No Source Control Provider.

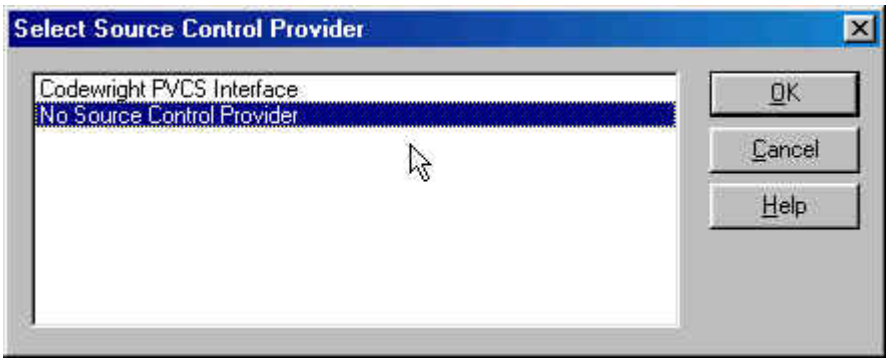

Figure 4. Select Source Control Provider Dialog

5.2 Source Control Tool Options

The Source Control functionality of Code Composer Studio offers standard features that are common with the popular Source Control tools in use. The selections include: prompt for options on get latest, …options on check-in, …options on check-out, …undo check-out, …on add, …on remove, …on show difference and so on. You are at liberty to choose those options you require or must use as per company policy. This option is displayed when you choose Project→Source Control→Options from the Code Composer Studio Project menu:

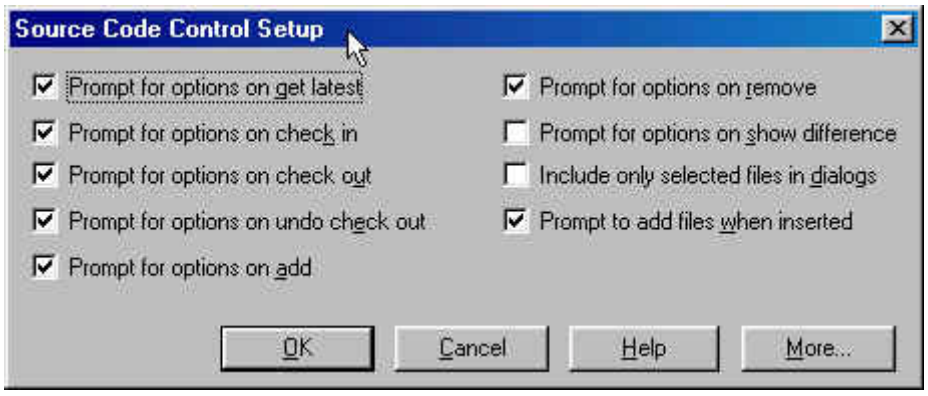

Figure 5. Source Control Setup Options

6 A Simple Project With Code Composer Studio 2.0 – Code Phase

Once you have completed configuring the Code Composer Studio environment for initial use in terms of setting up which external editor and what source control tool to use, you are ready to begin creating or coding your first project.

For the purposes of this application note, the supplied files and project from the HELLO2 tutorial are used. However, you can create your own code as well.

In order to begin, launch the Code Composer Studio software and go to the Project menu and choose the New option. The Project Creation dialog box appears and requests that you type in a project name. For demonstration purposes, the name used was Hello. The IDE automatically adds the .pjt extension to signify that it is a project file. The full name of this project is: Hello.pjt. This file stores all the settings for this project and references the various files that are used by this project.

Now that the Hello2 project environment is setup, you need to add source files to the project. Go to the File menu and choose the New option. If the external editor is functioning correctly, you should see the Codewright editor window open on top of the Code Composer Studio project window. At this point, you can either copy the contents of the HELLO.C file from the following default installation directory: C:\ti\tutorial\sim64xx\hello2, into the Codewright editor screen or write your own C source code in the same screen. You could also choose File→Open from Codewright and open the HELLO.C file.

You need to follow the above procedure for the following files in order to successfully build this project: VECTORS.ASM and HELLO.CMD. They can be found in the following default directory: C:\ti\tutorial\sim64xx\hello2. Clicking on the File→New menu item displays a dialog that allows you to choose whether the new file is a source file, a DSP/BIOS configuration file, a Visual Linker recipe or an Active X document. The last three are discussed later in this text.

6.1 Adding Files to Your Project

Once the procedure detailed in section 6 is completed, you can begin to add the source and other files needed to build a project, generate an executable, and load it onto the DSP target.

Code Composer Studio has provided you with multiple ways to add files to your project. Select any of the following methods:

From the Project menu, select Add Files to Project

From the Project Manager, right-click on the name of the .pjt file

Both these options display the Add Files to Project dialog box that allows you to locate or path out to the directory the required files are stored in are shown in Figure 6. This dialog also comes with a pull-down selection menu for Files of Type, so you can select files with different extensions such as C files, C++ files, assembly source files, library files or all file types that a user may wish to use in a project.

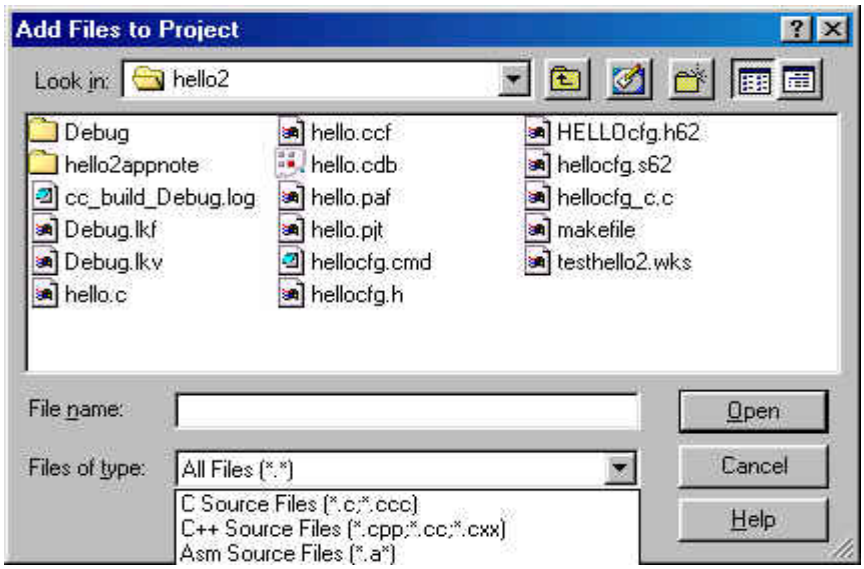

Figure 6. Add Files to Project Dialog

You need to ensure that the following files are added to the project before the executable can be built: hellocfg.cmd, hello.c, rts6400.lib. Once these files have been added, you can right-click on the project name in the Project Manager and Scan All Dependencies. This automatically adds the last files needed, the header files (*.h).

Once all the above files have been added to the project, you can check the Code Composer Studio Project Manager to review the file structure and examine what files were added to a particular folder. The Project Manager also supports drag and drop functionality from the Windows Explorer. In this example, the Project Manager should look like Figure 7:

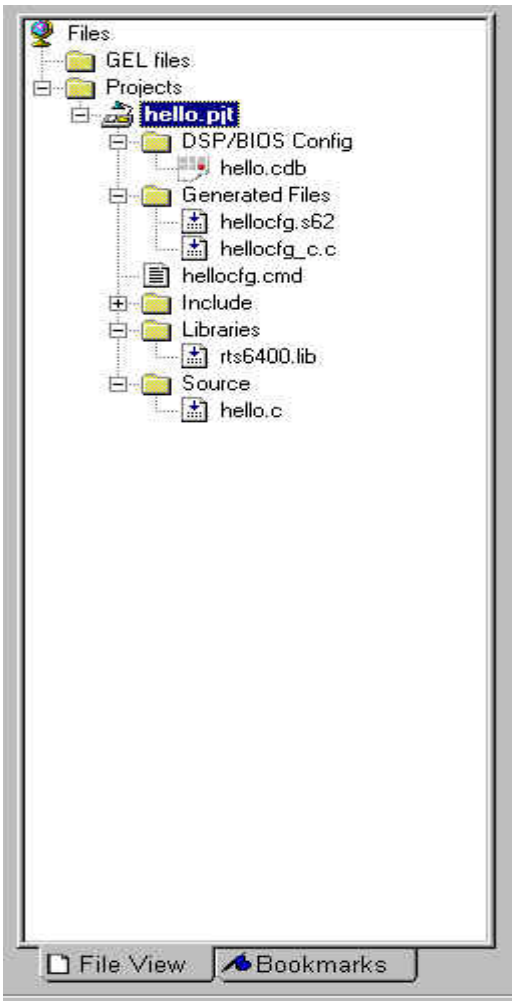

Figure 7. Project Manager With Project Files

The Code Composer Studio Project Manager also offers the capability of placing *bookmarks* in your code. A bookmark can be set at any line of code and is used to find and maintain important locations within the source code. Any bookmarks are saved in a Code Composer Studio workspace so they can be recalled at any time. Beside each folder name there may be a + sign. These signs can be clicked on to expand or open the folder to view all the files contained in it.

6.2 Description of the Project Files

Once you have placed all the source, library, linker, and header files into your project, you are ready to build your first executable. The following files were added to your project in section 6.1:

Hello.c (Source File) – contains all the C code that you will compile to form an object file.

.ASM files (assembly source files) – contains all the assembly code used to compile an object file.

RTS6400.lib – (library file) – is the run-time-support object library for use with C/C++ code. It contains the following:

ANSI C/C++ standard library

C I/O library

Low Level support functions that provide Input/Output to the host system

Intrinsic arithmetic routines

System startup routine, _c_int00

Functions and Macros that allow C/C++ to access certain instructions

*.h – (header files) – contain variable declarations and function definitions. These files are automatically added to your project if referenced in your C source file. Others are created as a result of using DSP/BIOS. This is explained in more detail later.

*.cmd – (linker command file) – contains linker options and names the input files for the linker. It also defines memory on the DSP and tells your code what areas it can write to.

Hellocfg.cmd – is the DSP/BIOS linker command file

Hellocfg.h – includes DSP/BIOS module header files and declares external variables for objects that are created in the configuration file.

Hellocfg_c.c – contains code for the CSL (chip Support Library) structures and settings, and is generated by the DSP/BIOS Configuration tool. It is recommended that this file is not modified.

Hellocfg.s62 – is an assembly language source file for DSP/BIOS settings.

Hellocfg.h62 – is an assembly language header file that is included by Hellocfg.s62.

Hello.cdb – is a DSP/BIOS configuration database file which stores the DSP/BIOS configuration settings.

GEL files – are .gel extension files that are either supplied with Code Composer Studio or are created by you. When added to a project, they can be used for customizing the environment, automating testing, configuring the DSPs' memory and registers and creating dialog and slider bars.

HELLO.CCF – exists in the project directory. This file is temporary and only exists when the configuration file is open. It is strongly recommended that this file is not modified, opened or deleted.

TIP – It is strongly recommended that the HELLO.CFG file is not opened, modified or deleted.

6.3 The Code Composer Studio Project Manager

The Code Composer Studio Project Manager for version 2.0 offers many features that are designed to make life easier for the DSP developer. This section provides some details on these new features.

The Project Manager offers the capability to work with multiple projects where many projects within the workspace are open, but only one project is active at a time.

The Project Manager provides you with the ability to save a currently active project, using the Project→Save command, before moving on to another project within the workspace.

Multiple configurations are supported with the Project Manager in that you are permitted to utilize separate build options for each configuration. You can also create Release or Debug (created by default) configurations and are free to add custom configurations. These project configurations define sets of project level build options and when specified, they apply to every file in a particular project.

Code Composer Studio also allows you to specify the order in which all the files in your project are to be linked. In other words, this functionality allows you to control the order that object files and libraries are linked at build time.

More information is available on the Project Manager topics in the Code Composer Studio online help.

6.4 Code Composer Studio and GEL Files

Code Composer Studio comes with its own scripting language that is called GEL, which is a subset of the C programming language. GEL stands for General Extension Language and can be used for many different functions in the Code Composer Studio environment in order to extend its functionality and capabilities. Any GEL file has the .gel extension appended to the end of the file. Currently, the software comes with about 35 built in functions that enable you to control the state of the target, access and control target memory, display results in an output window, and create dialog/slider boxes. You can also create your own GEL files to initialize and control the target DSP, for menu items, to have a particular GEL file run when Code Composer Studio is started or when a GEL menu item is selected, to setup a target memory map a certain way, and so on. These GEL files are completely portable and reusable (provided any target or memory map information is consistent) and very easy to maintain. You can also use GEL to customize your workspace and automate testing. GEL functions can also be added to the Watch Window so that they execute at every breakpoint. The GEL scripting language supports the following: function definitions and parameters, calling GEL functions, return and if-else statements, while statements, GEL comments, and preprocessing statements. Please refer to the Code Composer Studio online help for more detailed information.

7 Building a Project With Code Composer Studio 2.0

At this point, all the required files have been added to the HELLO project that are needed to generate an executable. Code Composer Studio saves any changes to the project setup as you make them, so if you closed the software after loading your files into the project – all you need to do is open Code Composer Studio and do a Project→Open. You also need to go to the File menu and issue a Load GEL command, which displays a dialog box that allows you to path out to where the GEL files are stored. In this case, the GEL files are stored in the following directory: C:\ti\cc\gel\init64xxsim.gel

This GEL file is needed and should be run in order to clear any breakpoints and to setup the EMIF (External Memory Interface) registers so that you can take advantage of this memory.

If you are using an actual hardware target, you may want to rest the target so that the DSP and its registers are ready to go to work. This can be done in a couple of different ways. First, go to the Debug menu and click on Reset DSP; this issues a software reset to the DSP (or simulated DSP). The other ways to reset the DSP are to either toggle the power to the target, or to use a third party reset utility. If you get an error message that Code Composer Studio cannot initialize the target, then you must reset the DSP target in order to re-establish communication with the hardware.

Once all the files are loaded, you can check the Load Program After Build checkbox in the Customize dialog that was described in section 4.3. This ensures that the executable is loaded immediately after a successful build. Please ensure that you have loaded and run the correct GEL file to setup the DSP target to receive your executable. In this case, the init64xxsim.gel file must be loaded and run by clicking on the GEL menu item and then the Resets→ClearBreakPts_Reset_EMIFset.

There are three ways to issue a Build command, from the menu or from the icon and they are as follows: select Project→Build, click the icon beside the active configuration dialog box, or right-click on the project name in the Project Manager (in this case, HELLO.PJT). Once you have chosen your preferred method to build, a Build window opens in the lower section of the Code Composer Studio interface. It will look like that shown in Figure 8.

Any errors and unresolved dependencies will appear in this window in red text. Provided you have all source files relating to this project loaded, double-clicking on any line containing an error in the build window causes one of two actions: if using the Code Composer Studio editor, the source file automatically opens up with the cursor flashing at the offending line **or**, if you are using an external editor, it becomes active in the task bar at the bottom of your PC screen and has the cursor flashing at the offending line of code that needs to be corrected. You also have the option to turn off the external editor by clicking on the icon at the top right corner of the icon bar within the Code Composer Studio environment. This allows you to selectively use one editor or the other or to toggle back and forth between editors.

Figure 8. Build Window Output

Your build window may not look exactly like the one in Figure 8 because your directory structure may be different than the test system that was used for the purposes of this application note. Double-clicking on any line in red in the build window opens the source file associated with the error in either your external editor or the Code Composer Studio editor and takes you to the suspect line of code. Errors are something that must be addressed before you can proceed with your development, but warnings are something that you can choose to ignore because they will not impede your progress.

7.1 Code Composer Studio Build Options

Code Composer Studio allows you to control all the aspects of the build process by incorporating the ability to customize the build options that you choose to use. You can set project-level options as well as file-specific options. The difference between these two levels of options is how they effect files. Project-level options are applied globally to all the files in the project. File-specific options apply specifically to the optimization of individual source files and override project level options. The reasons that you may wish to apply different build options to different files within the project are, for example, if you would like to build with more optimization on one file than another or to build with file size being more important than speed or vice versa.

TIP – If a custom build requires launching TI Code Generation Tools (with use of an External Editor, for example), the batch file DosRun.bat must be run in order to setup proper path and environment variables. It is advised that you run this batch file prior to doing a build. This file is located in the C:\ti directory if the default installation was done.

The Code Composer Studio Build Options provide you with the ability to control build order for the project, build steps for an individual file, the order that files are linked into the project, before and after build steps, and so on. The various tabs from the Code Composer Studio Build Options dialog are covered below. Figure 9 illustrates the actual Build Options dialog for C6000.

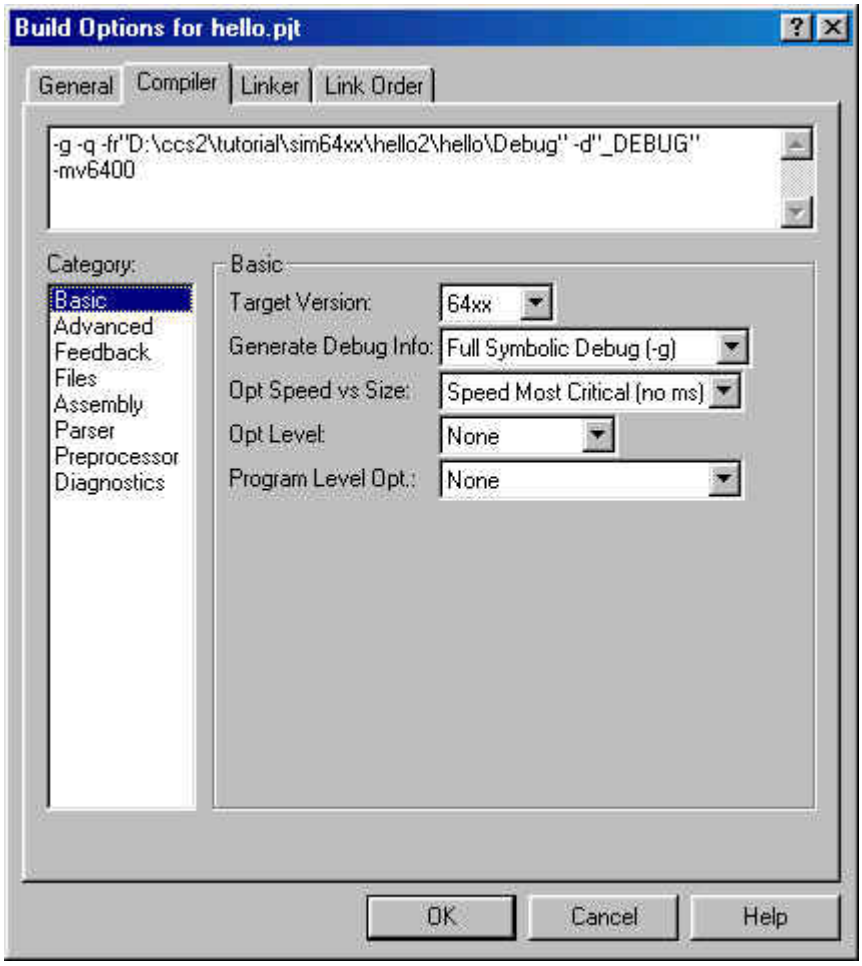

Figure 9. Build Options Dialog

As can be seen in Figure 9, there are four tabs on this dialog that provide a great deal of flexibility when building a project. These tabs each have your own purpose as is described below.

General Tab – This tab allows you to specify the order of the initial and final build steps, place additional steps in either the initial or final steps, change the order, apply custom build steps, delete any build step, exclude a particular step from the build process, remove any files that have been generated by the build, define a path and filename for the executable and any other files created during the build process, etc.

Compiler Tab – This tab is used to allow you to choose commonly used compiler options, optimization speed versus size, optimization level, and program-level optimization. The other categories provide you with the ability to choose endianness and memory model, control aliasing, provide feedback banners during the build, specify .obj and listing directories (among others), control the assembly process, control the parser and preprocessor functions, and select diagnostics. At the top of the Compiler options tab, there is a window (scrollable) that provides you with a line summary of all the options to be used during the final build. When checking or highlighting any particular category from Basic to Diagnostics, you are presented with various checkboxes, pull-down menus and/or text boxes. These features can be used with the default options or if preferred, you can select the options and functionality you prefer or need for a particular project. When any option is chosen, you can see the options change in the scrollable window at the top of the Build Options dialog. You can also simply add your own options to this scrollable window, but in many cases this requires a thorough knowledge of the actual text commands for the options requested. As can be seen in the scrollable window, the example string provides the path of the project .obj files, the configuration of the project (in this case, debug) and target information. You can also see that this project will be compiled with full symbolic debug (–g option), with no banners (–q option) and with symbols defined (–d option).

Linker Tab – Similarly to the Compiler tab window at the top of the screen, the Linker tab also provides a scrollable window that provides you with a summary of all the options that will be used during the link phase of the build. As described for the Compiler tab, whenever an option is chosen on the Linker tab, you can see the line summary or string change in the scrollable window. This scrollable window supports not only direct editing, but the ability to specify options that are not listed in the dialog. The executable filename and the directory of the executable are also present in this scrollable window.

TIP – Whenever options are typed directly into the scrollable window in either the Compiler tab or the Linker tab, you must click inside the window in order to update the display.

Link Order Tab – This tab allows you to choose the order that object and library files are linked into the project during the build. If you have implemented multiple versions of the same function in different files, link order is very important to ensure that the correct version is linked into the program being built. This tab allows you to specify the link order, change the link order, remove a file from the list of files to be linked, and so on. In the lower window on this tab, you are presented with a list of the files that are not in any order to be linked. This list of files will execute after the files in the link order window.

For more information on any of the Build Option tabs or categories, please refer to the Code Composer Studio online help.

As well as using the text linker through the Linker tab, you can also use the Visual Linker that is supplied with Code Composer Studio. The difference between these two methods is that the text linker generates a single COFF (Common Object File Format) object module from the object files of a particular project. The linker instructions in the linker command file (.cmd) enable you to unite object file sections and to attach sections or symbols to addresses or memory ranges. You can also define or redefine global symbols within the project.

Where the text linker combines the object files from a project to generate a COFF object module or executable file, the Visual Linker allows interaction from you when linking to create an executable. This method uses object files and a memory description as input. You can then drag and drop the object files to arrange a graphical model of memory. When you have chosen a satisfactory memory model, the executable (.OUT) can be generated.

7.2 Using the Visual Linker

The first step in using the Visual Linker functionality is to generate a *recipe* (.rcp) file. This recipe file tells us how to link an application and contains input files, hardware descriptions, and instructions on how to combine them.

You must first tell Code Composer Studio that the text linker is no longer desired and the Visual Linker is to be used from here on in. When the Visual Linker is chosen via the Linker configuration tool, it is selected at IDE level, not just at project level. This means that the Visual Linker will be used for every project for this particular target, not just for this one project. This setup remains in effect until you reopen the Linker Configuration tool and deselect this option. In short, the Visual Linker translates the linker command file (.cmd) from the text linker into a recipe (.rcp) file for use with the Visual Linker.

In order to generate the recipe file, you need to go into the Code Composer Studio Tools menu and select the Linker Configuration option. That presents you with the dialog box shown in Figure 10.

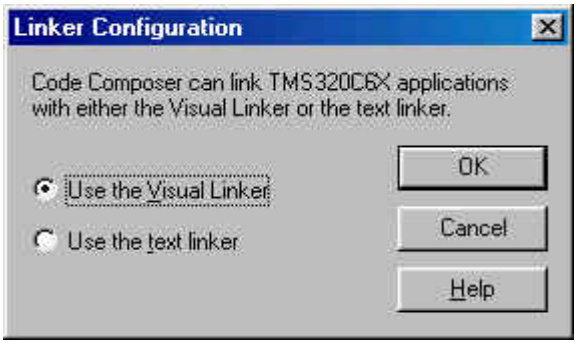

Figure 10. Linker Configuration Dialog

From the Linker Configuration dialog, you can select whether to use the Visual Linker or the legacy text linker. Once the option for the Visual Linker has been selected, simply click OK and then select Project→Rebuild All from within the Code Composer Studio environment. Code Composer Studio then proceeds in going through the build process. Please bear in mind that the project does not progress beyond the link stage due to the fact that the Visual Linker has been selected.

Once the software has progressed to the link stage, you are presented with a series of messages as detailed in Figure 11.

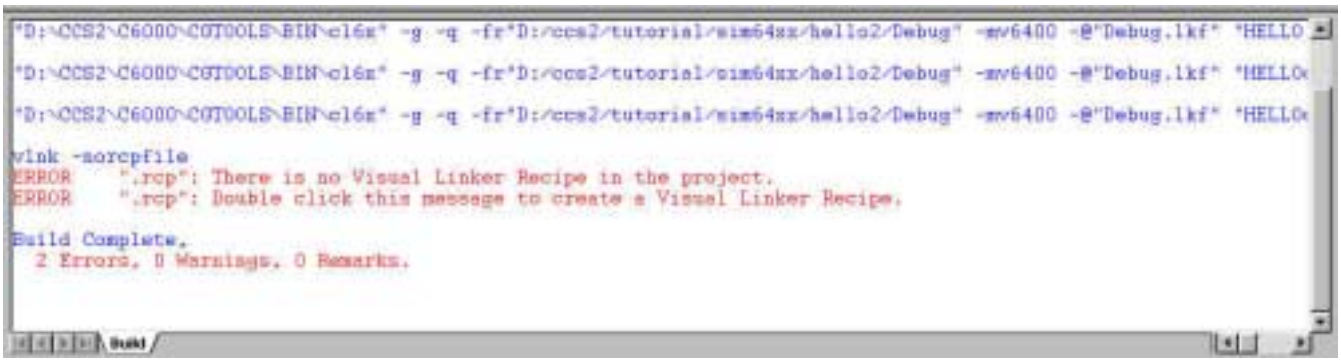

Figure 11. Build Window Utilizing Visual Linker

Code Composer Studio has now informed you that there is no visual Linker recipe file in this project. In order to generate one, simply double-click on the lower of the error messages as it says.

Once you have done this, Code Composer Studio presents a Visual Linker Wizard to walk you through the process. There are a total of nine steps to this process that walk you through it.

During this process, you have the flexibility to choose when to initialize global variables, to do a full or partial link, and then present a summary of the features of the recipe that is about to be created. You can then choose to accept these features or continue on through the Wizard in order to change them.

The Wizard next prompts you to translate the SECTIONS directives and provides several options for you to choose. You also have the option not to translate these directives.

Following the options for SECTIONS directives, you are provided with output file information and the opportunity to change the name and/or location of both the output file and the map file from the ones Code Composer Studio has detected.

Moving forward through the Wizard, you can choose to accept the default sizes for the stack and heap size, or change them to increase or decrease the sizes.

In step 7 of the Wizard, Code Composer Studio has detected a code entry point and section for interrupt vectors. Here, you have the choice of accepting what Code Composer Studio has detected or to change these options.

The next step involving the Wizard is the Roots portion. This page provides a list of all the input sections that are not reachable from the noted code entry point or interrupt vector section. Any section listed here can be included in the linked image by checking the appropriate box beside the section.

The final step of the Wizard is some information about the Mission of the Wizard. It informs you that the recipe file has been added to the project and can be found in the Code Composer Studio Project Manager. The name of this file will be HELLO.RCP. In order to use this file, you only need to double-click on it and the screen in Figure 12 should appear.

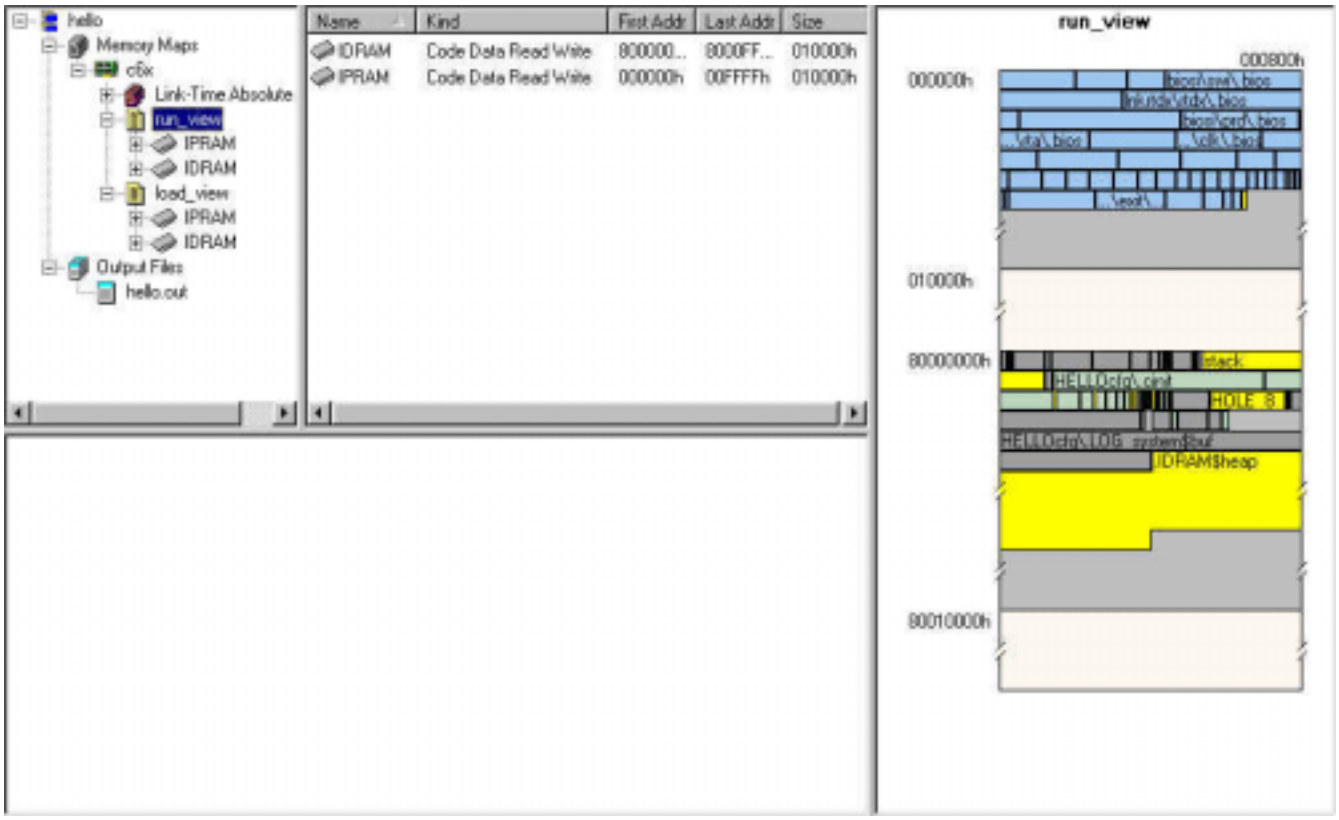

Figure 12. Visual Linker Recipe Screen

The screen in Figure 12 indicates that a recipe file has been successfully created. You can resize the window so that all four panes are visible. Depending on the target configuration, the elements contained in this file may vary and may not be exactly as depicted in Figure 12.

If you want to accept the recipe that the Visual Linker has generated, then continue with building your code. However, if you want to obtain more information on the specifics and particulars of using the Visual Linker or how to allocate or change a program in memory, it is strongly recommended that you refer to the online help.

8 Code Composer Studio 2.0 and the Debug Phase

A number of features have been added and/or improved upon in this version of Code Composer Studio. Those who have grown accustomed to the Watch Window in previous versions of Code Composer Studio will note that the new Watch Window provides more functionality and more features. These new features and improved functionality assist you during the debug phase of the code development cycle by allowing you to watch local variables and to freeze updates.

Texas Instruments has also added a Symbol Browser to the environment and this feature provides information on files, labels, and variables of a loaded project and executable.

Code Composer Studio 2.0 also provides C++ support for those who prefer to work with that language to develop and debug target code.

More information on the new features and functionality is provided in section 8.1.

8.1 Using the New Watch Window

When you are writing and testing programs, there are often cases where you will need to check/verify the value of a variable while your program is executing. A good way to do this is to use the Code Composer Studio Watch Window.

In order to use the Watch Window, you need to ensure that Code Composer Studio has been launched and that your executable has been launched on the target. In this case, you open the HELLO2 tutorial and load the executable file, HELLO.OUT. Once this is done, you set a breakpoint at the line: LOG_printf(&trace, hello world!);

The breakpoint can be set using either a double-click in the selection margin or by right-clicking on the line of code you wish to set the breakpoint at, then selecting the Toggle Breakpoint option. Breakpoints can also be set using the F9 key or by using the icon that looks like an open hand.

At this point, you can either use the different colored icons in the selection margin to indicate breakpoints, Probe Points or Profile Points or use the Customize dialog box to choose the option of the whole line to change color wherever a breakpoint, Probe Point or Profile Point is set. You can even get creative and select any color you want to indicate the various types of breakpoints in the selection margin.

TIP – Please ensure that whatever color you choose does not obscure the keywords in the editor because colors similar to your keywords makes seeing them difficult.

You then need to open the Watch Window by going to the Code Composer Studio View menu and selecting the Watch Window option. The Watch Window opens at the bottom-right corner of the Code Composer Studio environment. This area shows the value of watched variables when running your code. The default mode of the Watch Window displays the Local variables tab to display variables local to the function being executed.

In order to add a variable or variables to the Watch Window, several methods have been implemented to achieve this. You can click on the expression icon in the Name field of the Watch Window and simply type the name of the variable in this field or you can highlight the variable in the source window, right-click and choose the Add to Watch Window option.

You can also use the Watch Window to watch structures, and by double-clicking on any element in the structure, the value for that element can be edited. The same principle applies with arrays. Please note that whenever a variable is changed, the change is reflected in the Watch Window and the value is in red. The red color of the variables indicates that the value has changed most recently.

8.2 Working With the Symbol Browser

The Symbol Browser can be used to look at and review files, functions, labels, globals and types of a loaded COFF or executable (.out) file. The Symbol Browser is invoked by going to the Code Composer Studio Tools menu and then down to the Symbol Browser option. There are five tabs in the Symbol Browser, each connected to a different window. Each of these windows displays information on files, functions, labels, globals, and types. Each window has information that represents various symbols and if a plus (+) sign precedes this information, that means that the information or node can be expanded.

The Files tab displays a list of all the files that are in the currently loaded executable. If you click on the plus (+) sign in front of the executable filename, a complete list of all the files that are in the output file is presented. Each file listed can be further expanded to show all the functions in that file and each function can be expanded to show the local variables in the function and so on. Double-clicking on a filename displays the files' source code in a document window. This action holds true for the other tabs within the Symbol Browser as well.

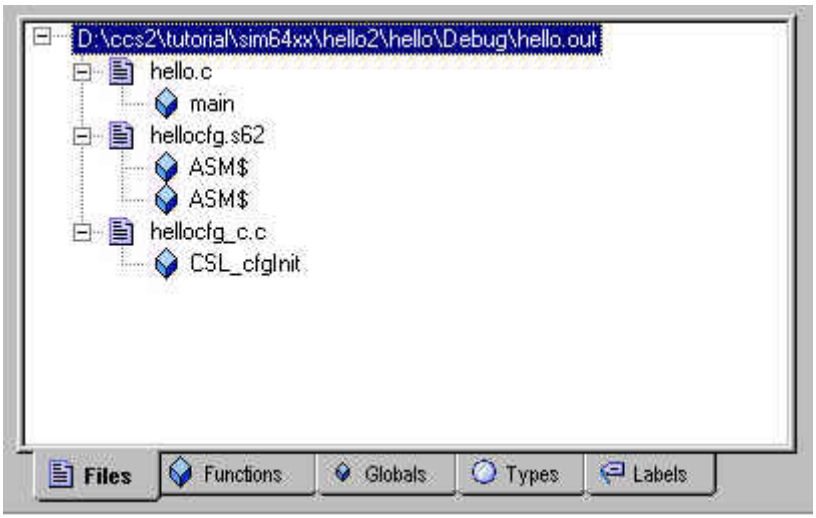

Figure 13. Symbol Browser Window

8.3 C++ Support With Code Composer Studio 2.0

The latest version of Code Generation Tools and Code Composer Studio 2.0 allows you to write and debug C++ code using standard debug tools. The Code Composer Studio IDE and the Code Generation Tools have both been enhanced to support C++ project build options.

There are some limitations when using GEL expressions with C++ and they are summarized as follows:

- No pointer adjustment is made when casting to or from a virtual base class
- Identifiers after a "." Or "–>" operator cannot be qualified
- The destructor of a class cannot be named
- Pointer to member values appear as implementation structures
- Type synonyms defined with typedef are not available
- Types must be fully qualified and cannot be partly qualified the same way as identifiers
- The IDE does not identify dynamic or run-time type objects that base class pointers refer to
- GEL expressions must dereference C++ references, otherwise they appear as pointers

It is recommended that you be familiar with C++ and that the online help be reviewed before moving forward in working with C++.

9 Code Composer Studio and the Analyze Phase

You are now ready to begin the analyze phase of the code development cycle. In this phase, you get information from the system returned through specific analysis tools. This information allows you to monitor and measure the performance of a particular program, in this case the HELLO2 tutorial.

The tools demonstrated in this section are DSP/BIOS, RTA and Advanced Event Triggering (AET). Using DSP/BIOS allows you to view the status and performance of a particular program in real-time. By having access to this information, you can debug and optimize your system. Any project that has a DSP/BIOS configuration file can use these analysis tools.

Advanced Event Triggering consists of several tools that add to your ability to debug target code more quickly. The tools that comprise Advanced Event Triggering are: Event Analysis and the Event Sequencer.

Event Analysis assists you to set up regular hardware debug functionality while the Event Sequencer lets you search for certain conditions to occur in your executable and to cause specific actions to occur when those conditions are met.

Code Composer Studio also allows you to view your target code visually by graphing your code as it progresses. This is beneficial as you can detect or inject noise and actually view what the algorithm is doing as it progresses. The IDE supports the following types of graphs: eye diagrams, FFT plots, image displays, time/frequency graphs, and constellation plots. Code Composer Studio also allows you to inject or extract data at any point in the algorithm and copy it to or from a file.

9.1 Using DSP/BIOS With Code Composer Studio 2.0

DSP/BIOS can be used on any application that needs to be probed, traced or monitored in real-time. Each module of DSP/BIOS can be linked into an application as required and only those referenced in the application will be called.

DSP/BIOS consists of the following components: DSP/BIOS Configuration Tool, DSP/BIOS Real-Time Analysis Tools and the DSP/BIOS APIs. For more information on any of these components, please refer to the Code Composer Studio online help.

In order to proceed with using DSP/BIOS, you must ensure that Code Composer Studio is open and that your project is loaded. You must also ensure the file STD.H is loaded in order to utilize the DSP/BIOS API and any other header or .h files for modules your program may use. You must also include a DSP/BIOS configuration file included with your project in order to take advantage of the DSP/BIOS API. This configuration file defines all objects and properties that will be used by the application program. Go to the File→New menu and choose the DSP/BIOS Configuration option. At this point, you are presented with the window in Figure 14:

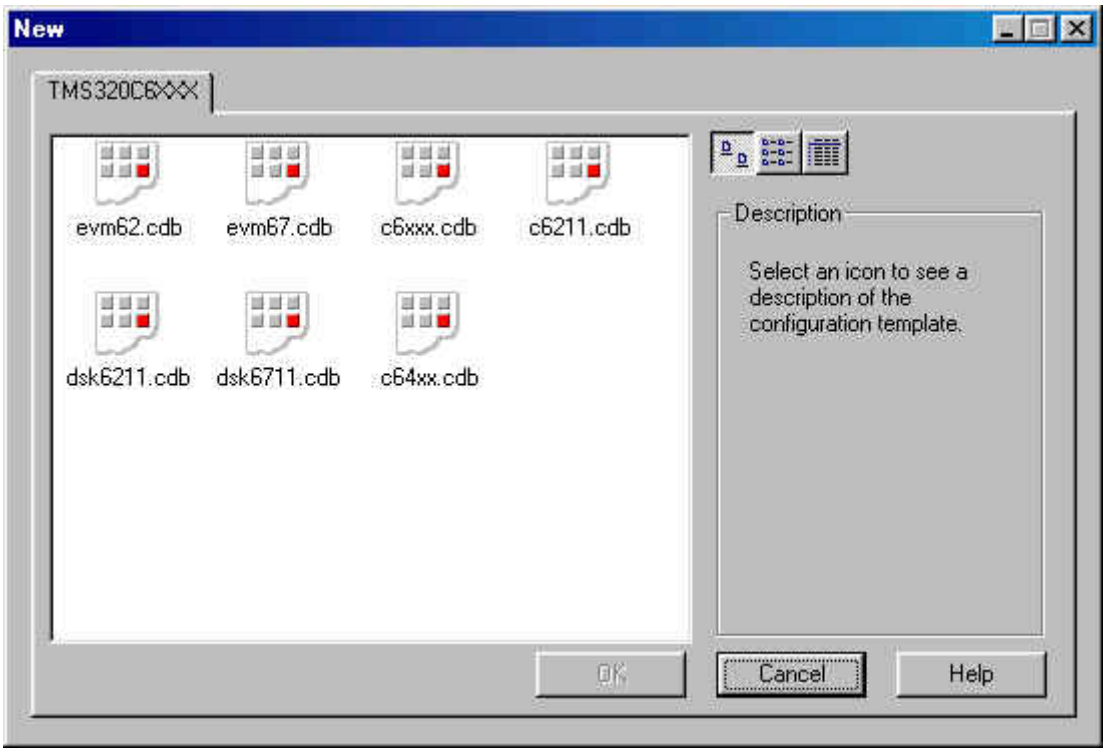

Figure 14. DSP/BIOS Configuration Template Window

You are required to select the configuration database (.cdb) file template that most closely matches your target. In this case, the C64x.cdb template is used. Once this has been accomplished, please note that the Configuration window is now visible on the screen. On the left side of this screen, you see categories of modules and if you click on the plus (+) sign beside any of these categories, you are presented with the modules contained in each of these categories. You can also see the various module managers and if you click on the plus (+) sign to the left of the module manager, you can see the modules objects. The right side of this window displays the properties of the selected module manager or object.

For this exercise, you need to create a DSP/BIOS configuration file, so from the information you currently have on the screen, make the required changes as outlined below:

- From the Event Log Manager option, LOG – please choose Insert Log after right-clicking on this item
- right-click on the newly formed LOG0 object and choose to rename it to trace
- From the Code Composer Studio menu, choose File→Save and save this file as HELLO.CDB

This file should look like Figure 15:

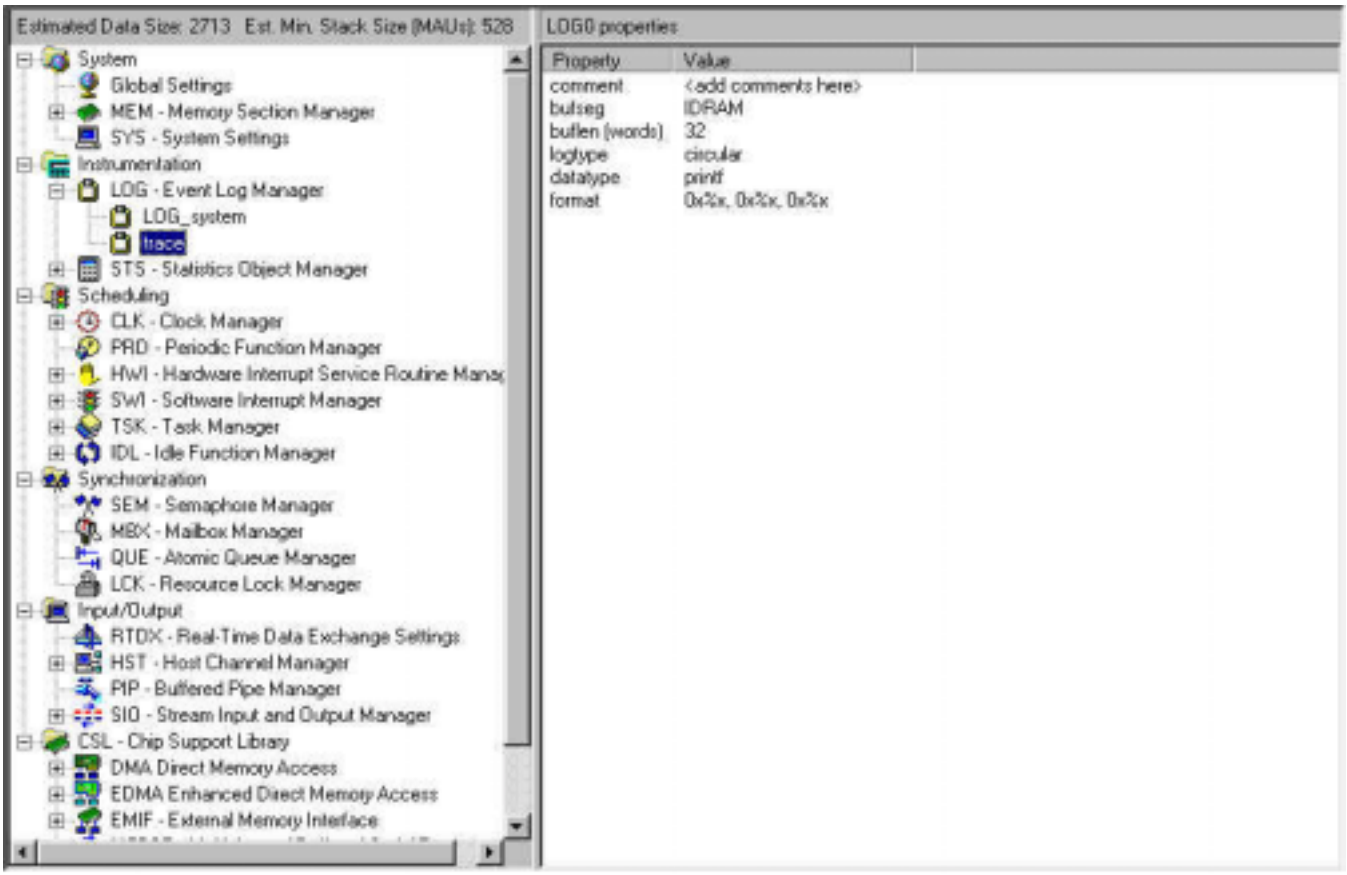

Figure 15. DSP/BIOS Configuration File

Saving the configuration in Figure 15 generates a series of files. These files are header files, linker command files, BIOS header files, and so on. You also need to remove the library file that you used previously. A file named hello.ccf, which is a temporary file that is spawned when opening the configuration file, is shown. It is not recommended to open, modify or delete this file.

You then need to add the specific DSP/BIOS files to the project as you did in the beginning. Once these DSP/BIOS generated files have been added to the project, you need to remove the files they have replaced. In this case, the files hello.cdb and hello.cfg.cmd, are added or need to be added to the project. Once you have added these files, ensure you are also working with the Visual Linker or the text linker as the case may be, and go to Project→Rebuild All.

Please ensure that the executable has been loaded after the Rebuild All step and then go to the DSP/BIOS message log. In order to launch this, simply go to the DSP/BIOS menu item and down to the Message Log option. This step presents the message log at the bottom of your code Composer Studio screen. In order to test the program, right-click on the message log screen and choose Properties Page. The Message Log Properties dialog appears. At this point, you need to check the Log to File checkbox and click OK. Go to the Debug menu and click on the Run option. After a few seconds, the HELLO WORLD string should appear in the message log window. Right-click on the message log window and then do a File→Open and in the Debug folder, there should be a HELLO.TXT file. If you open up this file, it should contain the same message that was just in the Message Log window. If you don’t see anything in the HELLO.TXT file, it is likely because you did not close the Message Log window.

At this time, it is also recommended that you disable the RTDX functionality, because Profiling is used in section 10.1. In order to do this, go to Tools→RTDX→Configuration Control in the Code Composer Studio environment. This should be done because on most DSP platforms, the profiler and RTDX cannot be used at the same time. For more detailed information on moving forward with DSP/BIOS and the Real Time Analysis Tools, it is recommended that you refer to the Code Composer Studio tutorial and online help.

There is also the new functionality of the Chip Support Library (CSL) which is a C language interface for configuring and controlling peripherals that are on chip. These peripherals include timers, DMA, and McBSP. It consists of modules that are built and archived into a library file. More information is available on this topic in the online help.

9.2 Advanced Event Triggering (AET)

Advanced Event Triggering or AET consists of the following components: Event Sequencer and the Event Analyzer. Both of these components add to your ability to debug and analyze your DSP code.

Event Analysis assists you to configure hardware debug tasks called jobs. You can set breakpoints, action points and counters by right-clicking or by drag and drop.

Event Sequencing allows you to check for certain conditions to be met in a target DSP program and to trigger actions to occur when those specific conditions are met.

In order to setup a job with the Event Analysis component of Advanced Event Triggering, the target processor must contain and support on-chip analysis capabilities. Event Analysis can be selected from the Code Composer Studio Tools menu, under the Advanced Event Triggering option. The DSP simulator (C64x) used for this exercise does not currently support Advanced Event Triggering, so a brief description on how to setup this functionality is provided. For more detailed information, it is advised you refer to the Code Composer Studio tutorial and/or the online help. The Advanced Event Triggering tutorial has been designed to run with the C6211 DSK and C6711 DSK DSPs, so it is recommended that you have one available in order to work through the tutorial lessons.

If you right-click in the Event Analysis window and choose a particular job, the job menu is actively built. The job menu is dependent on the target DSP setup or configuration. Once enabled, a job performs analysis on your code when it is run on the target. If a job is not supported on any particular target, the job is grayed out and not accessible.

If you choose to select a job from a source file, you can select information in the Code Composer Studio Editor such as line, variables or code range and the Event Analysis tool assists you to fill in the job dialog.

You can view the job status by observing the icon in the Enable column. The following jobs can be performed using this tool: hardware breakpoints and hardware breakpoints with count, chained breakpoints, data and program actionpoints, watchpoints and watchpoints with data, data access counter, watchdog timer, count, and counter in range. The status of a job can be modified in two ways: right-clicking on an icon on the Event Analysis display or by selecting a job in the Event Analysis display, modifying it and then clicking on Apply.

In order to create an Event Sequencer program, you can create Global Actions, Global If statements, and states from the toolbar. These icons can be dragged and dropped from the Sequencer toolbar into the Sequence Program Editor window in order to create a sequencer program.

For more information on Advanced Event Triggering, please consult the online help and ideally, work through the Code Composer Studio tutorial.

9.3 Graphing With Code Composer Studio

Due to the fact that graphing is one of the original features of Code Composer Studio and this document is intended to illustrate the new features and functionality, little time is spent on these capabilities within Code Composer Studio.

This advanced signal analysis interface enables you to critically and thoroughly monitor signal data. The Code Composer Studio Graph menu provides a lot of flexibility through the many options that it offers. Code Composer Studio allows you to select from a variety of graph types such as: FFT, image, eye, constellation, and time/frequency.

In order to visually analyze your code, for simplicity, the volume1 tutorial that is supplied with Code Composer Studio is used. You need to open the project file volume.pjt and load the volume.out executable. When the executable is loaded, please issue a DEBUG RESTART and then a DEBUG GO MAIN to set the program counter to the main() routine, and set a breakpoint and a probepoint (using a right-click) on the following line in the volume.c file: processing(input, output);

Once you have done this, you need to go to the File menu and down to the File I/O option to configure the input to stream into the executable file you are about to graph. If you do not do this, you will get invalid waveforms in your graph. Please click on the Add File button and check the Wrap Around checkbox. Once you have clicked on the Add File button, you should see a File Input dialog and look for a file called SINE.DAT. Please select this file.

The File I/O dialog appears on screen with the File Input tab displayed. There are three fields underneath the File Input field as follows: Probe, Address and Length. The Probe field tells whether the point you wish to input data to is connected or not. The Address field identifies the start address to input or output the data to or from. The Length field tells us how many samples are transferred every time a Probe Point is reached.

The Address field accepts labels or numeric addresses. In this case, enter inp_buffer. For Length, please enter 0x64. then click on Add Probe Point. This displays the Break/Probe Point Manager dialog. You can click on the entry in the Probe Point field in order to highlight it and fill in the fields above the Probe Point field. Click on the Connect To pull down and select the File In ….sine.dat option. Click Replace and then OK. The File I/O dialog should show that this point is now Connected. Next you need to go to the View menu and to the Graph option and choose the Time Frequency graph. This displays a Properties dialog that allows you to select or change various graphing options. Name this graph Input , the start address is inp_buffer and the Acquisition Buffer and Display data size are set to 100 and click ok.

Once you begin to Animate your executable, you should see a perfect sine wave as illustrated in Figure 16.

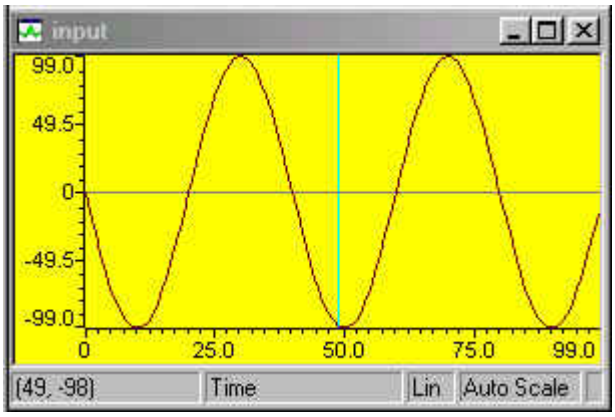

Figure 16. Graph Window

You can also add the inp_buffer variable to the Watch Window and view the data as it is being streamed into the code. This window appears as in Figure 17.

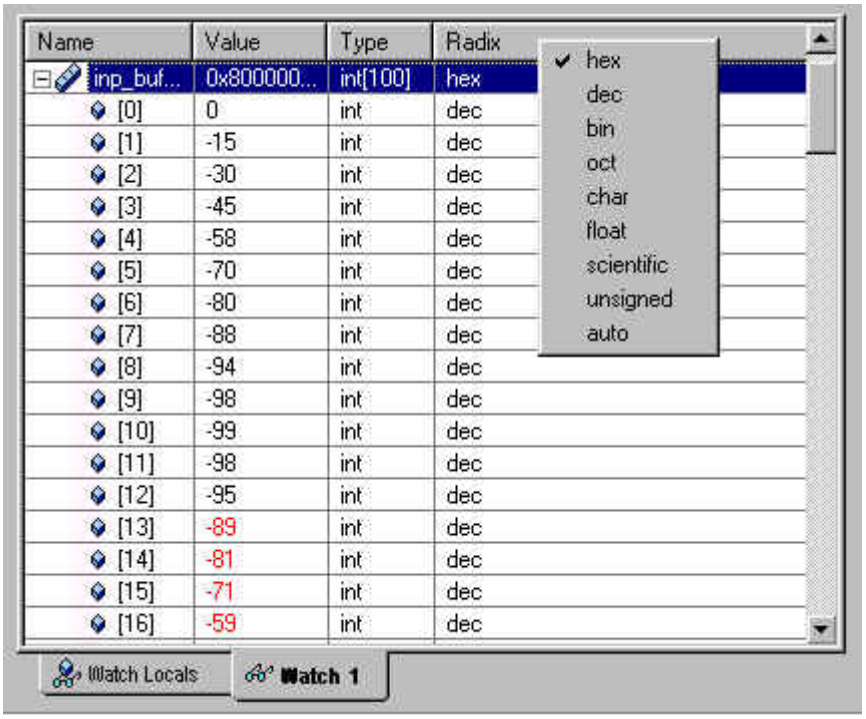

Figure 17. Watch Window

Note that by left-clicking on the RADIX column in the Watch Window, you can change or select the data type that you want to view.

10 Code Composer Studio and the Optimize Phase

10.1 Using the Profiler

In the optimize phase of the code development cycle, the goal is to profile or benchmark your code so that it is of small size and runs at optimal speed. By profiling the code, you can see where the most CPU cycles are being used and in turn, you can focus on these areas and attempt to optimize code performance and size. Code Composer Studio offers two methods of doing this: using the Code Composer Studio profiler and using a feature new to this release, the Profile Based Compiler or PBC.

The Code Composer Studio profiler analyzes your executable code and determines areas that are need of optimization. In order to begin using the Code Composer Studio profiler, you must keep several things in mind as follows:

- You must remember to **not** reset the profile clock between the start and end breakpoints, otherwise the profile statistics or results will not be correct. Start and end breakpoints are set by the profiler to determine a range to be profiled and in turn, optimized.
- You should be aware that more branches and function calls that are within an area to be profiled will cause the profiler to run slower. The profiler sets start and end breakpoints to define the profile range and the profiler also sets breakpoints before and after each branch and function call. Each of these breakpoints causes the processor to halt in order to be handled and this slows performance of the profiler.
- You also need to bear in mind that when profiling in ROM, hardware breakpoints need to be used. The number of breakpoints that can be set are dependent on the processor itself and as a result, you may not be able to profile a range of code that contains branches. There is a workaround to this issue in the online help.

In order to begin using the profiler, you must first have a project opened and an executable loaded onto the target. Once this has been done, go to the Profiler menu and choose the Enable Clock option. The other selections from the Profiler menu are: Clock Setup, View Clock and Start New Session.

Clock Setup – allows you to choose whether to count CPU cycles, branches or NOPs. You can also select the Instruction Cycle time in nanoseconds and select a Pipeline Adjustment.

View Clock – allows you to simply view the CLK variable.

Start New Session – presents you with a dialog with which to name a particular profile session. The default name is MySession.

When you have named your profile session and clicked OK, a Profile Statistics window opens at the bottom of the Code Composer Studio screen. It should look similar to Figure 18.

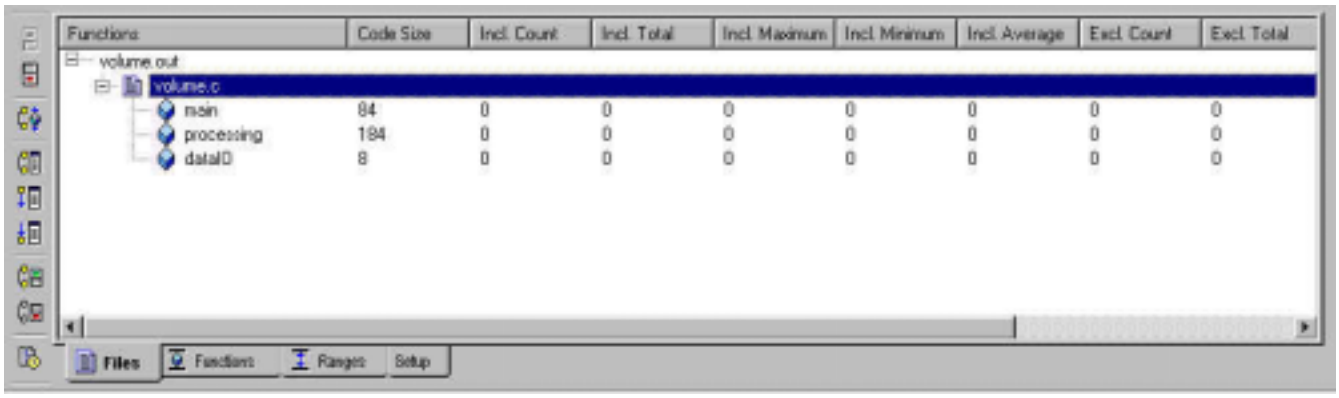

Figure 18. Profile Statistics Window

This window was actually too big to fit on the page for the purposes of this application note, but as can be seen, it provides the capability to calculate inclusive and exclusive counts, information on file size, information on functions, ranges, and setup. The setup tab provides information on the start and end points that you have chosen. The icons on the left side of the profile statistics window allow you to enable or disable a session, profile all the functions in your code, create a profile area (for profiling a few lines at a time), create start and end points of where to profile, and to enable or disable a profile area. Another feature of this window is the ability to generate reports on the profiling statistics after you have profiled your code.

For the purpose of this application note, the Profile All Functions option was selected from the Satistics window. A Profile Range has been set up in the left-hand selection margin of the C source code and that should look similar to that in Figure 19:

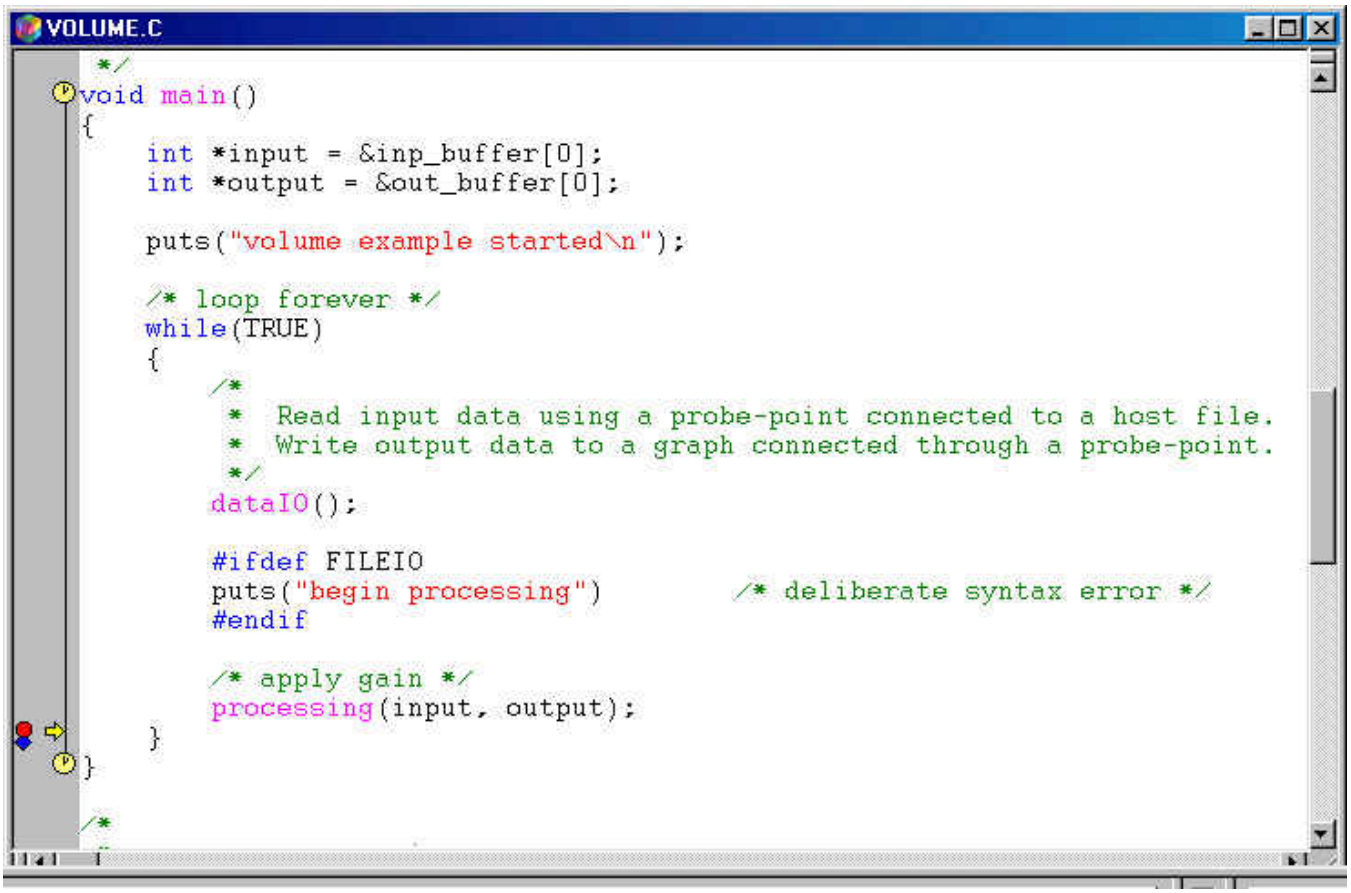

Figure 19. Profile Range Example

Once you begin to animate (Debug→Animate) the code, the various fields within the profiler stats window begin to populate with data obtained from the profiling process. From here, you can determine where the majority of the CPU cycles are being used, and can begin to optimize from there with the goal of executable speed or size.

10.2 Profile Based Compilation (PBC)

Code Composer Studio also incorporates some new profiling functionality and that is Profile Based Compilation. This feature allows you to choose between code size and code speed. PBC accomplishes this by building your application with various compiler options and executing each version, collecting profile statistic information as it goes. Once it has collected all this information from the various builds it has completed, PBC determines the best configurations and assignments of compiler options for each function. Then it is up to you to determine which configuration or assignment is best for your specific application.

In order to enable to the PBC, simply go to the PBC menu at the top of the screen and down to the Enable option. This displays the PBC Startup dialog that walks you through either the tutorial or the Getting Started phase of using this tool. Clicking OK opens up the PBC Wizard, which walks you through the process of creating configurations for your application. The PBC Wizard is illustrasted in Figure 20:

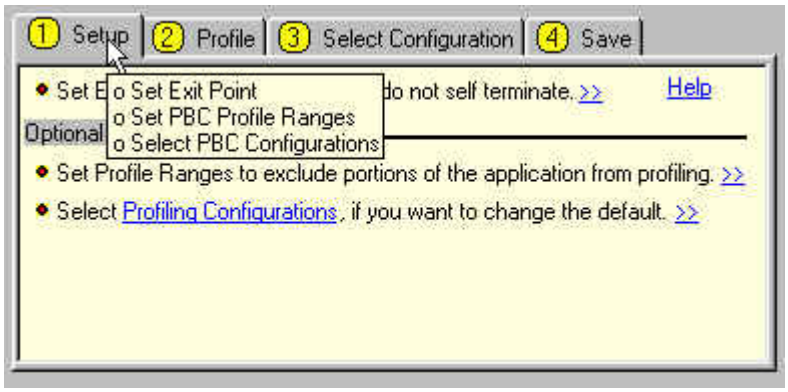

Figure 20. PBC Wizard Dialog

Each tab is numbered to allow you to follow the process through in logical progression. Passing the cursor over each tab displays options for each step in the process.

The Setup tab allows you to select Exit Points, Profile Ranges, and PBC Configurations.

The Profile tab provides the ability to Build and Profile a configuration, or to ReBuild and ReProfile a configuration.

The Select Configuration tab lets you view your selected configuration and view or apply any overrides. Overrides are specific compiler options that have been ignored for a particular function.

The Save tab lets you verify your configuration and save it to a project.

When building and profiling from the Profile tab, please ensure that you have created an exit point for any application you wish to profile that is not self-terminating, otherwise the build and profile stage could go on indefinitely. For more information on Profile Based Compilation, please refer to either the online help or the PBC Wizard to walk you through the process.

11 Summary

Over the past pages of this application note, many things were accomplished and many new features and functionality were introduced. In summary the following were discussed:

- The building of code using the Code Composer Studio Editor and Project Manager
- How to link a program with the Visual Linker and to configure CCS to use either the Visual Linker or the text linker
- How to set up CCS to use an external editor and source control
- The debug phase of the development cycle where the Symbol Browser and C++ Support were introduced
- Steps to analyze code, and DSP/BIOS, Chip Support Library, Advanced Event Triggering, and graphing functionality
- The CCS profiler and Profile Based Compilation

By now, you should have a basic understanding of the Code Composer Studio environment and what all the features included with the software do. Unfortunately, it is impossible to provide the detailed information in this application note to make one an expert with the software. That will come with exposure and practice with the environment as well as reading through the online help and working with the examples that are supplied.

12 Glossary of New Features in CCS 2.0

Chip Support Library – a C-language interface for configuring and controlling on-chip peripherals. Included are a set of macros for accessing specific registers and bit fields as well as eXpressDSP naming conventions. Texas Instruments has also included support for XBUS and PCI and have standardized the naming conventions for CSL header files to CSL_

Code Composer Studio Setup – has been enhanced to include many new target-specific configuration files to reflect the many new TI DSPs on the market.

Code Generation Tools – the Texas Instruments compiler has been enhanced to support C++. There is also a new linker option to enable function subsections and the run-time libraries support both C and C++.

CCS Debugger – has been upgraded so you can specify the search path for include files and to view local variables while debugging. There is also support for shared memory and a series of new GEL functions have been added.

DSP/BIOS – now supports C55x and C64x platforms as well as C++. There is also a HWI dispatcher that supports the C54x platform. The configuration and analysis tools have also been enhanced.

CCS Project Manager – has been enhanced to support multiple projects and allow you to save an active project before switching to another project. There is also support for multiple configurations and the ability to specify a link order when building a project. External make file support and source code control has also been added.

RTDX – Real Time Data Exchange has been added to allow you to transfer data between host and target without interfering with the target application. This allows you to monitor your system in real-time which is important for providing a realistic representation of your system in time critical applications.

Text Editor – has been upgraded to provide you with more flexibility for choosing cursor mode, editor properties, selection margin, keyword highlighting, choosing a third party editor, etc.

TI Command Window – many new features have been incorporated into this area and they are as follows: Profile Based Compiler, Watch Window, Symbol Browser, Component Manager, Update Advisor, and Advanced Event Triggering.

TMS320 Algorithm Standard – TI has created this algorithm standard to provide a common platform or template for any algorithms to run on TI DSPs. This allows these algorithms to be portable and standardized.

Visual Linker – allows you to visually see and configure your code into a graphic representation of memory on the DSP.

IMPORTANT NOTICE

Texas Instruments and its subsidiaries (TI) reserve the right to make changes to their products or to discontinue any product or service without notice, and advise customers to obtain the latest version of relevant information to verify, before placing orders, that information being relied on is current and complete. All products are sold subject to the terms and conditions of sale supplied at the time of order acknowledgment, including those pertaining to warranty, patent infringement, and limitation of liability.

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

Customers are responsible for their applications using TI components.

In order to minimize risks associated with the customer's applications, adequate design and operating safeguards must be provided by the customer to minimize inherent or procedural hazards.

TI assumes no liability for applications assistance or customer product design. TI does not warrant or represent that any license, either express or implied, is granted under any patent right, copyright, mask work right, or other intellectual property right of TI covering or relating to any combination, machine, or process in which such products or services might be or are used. TI's publication of information regarding any third party's products or services does not constitute TI's approval, license, warranty or endorsement thereof.

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. Representation or reproduction of this information with alteration voids all warranties provided for an associated TI product or service, is an unfair and deceptive business practice, and TI is not responsible nor liable for any such use.

Resale of TI's 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, is an unfair and deceptive business practice, and TI is not responsible nor liable for any such use.

Also see: Standard Terms and Conditions of Sale for Semiconductor Products. www.ti.com/sc/docs/stdterms.htm

Mailing Address:

Texas Instruments
Post Office Box 655303
Dallas, Texas 75265