Application Report
SPRA004 - October 2001

Code Composer Studio IDE v2 White Paper

John Stevenson *Texas Instruments Incorporated*

ABSTRACT

Designed for the Texas Instruments (TI) high performance TMS320C6000™(C6000™) and power efficient TMS320C5000™(C5000™) digital signal processor (DSP) platforms, the Code Composer Studio™ IDE is a development environment that tightly integrates the tools needed to create winning DSP applications.

Key components of the Code Composer Studio IDE include:

- Intelligent IDE including CodeMaestro™ technology
- C/C++ Compiler, Assembly Optimizer and Linker (Code Generation Tools)
- Real-Time Operating System (DSP/BIOS™)
- Real-Time Data Exchange between host and target (RTDX™)
- Update Advisor
- Instruction Set Simulator
- Advanced Event Triggering
- Data Visualization

The Code Composer Studio IDE integrates all host and target tools in a unified environment. It also simplifies DSP system configuration and application design to help designers get started faster than ever before.

Supported Operating Systems:

Windows 98
Windows 98 Second Edition
Windows NT (service pack 4 or higher)
Windows 2000 (service pack 1 or higher)

Contents

List of Figures

1 Introduction

With the emergence of new converging products that feature wireless communication, speech recognition, multimedia, and Internet telephony all in the same device, designers are relying on DSPs to provide the computing horsepower necessary for crucial real-time performance. Programmable DSPs provide software engineers with the ability to reduce time-to-market while still providing an optimized solution for the application at hand. But effective software is required to fully harness the power of DSPs.

The size of development teams working on embedded projects is growing dramatically as companies drive to produce products that integrate several technologies. This, combined with today’s acquisition strategies, serves to produce development teams that may be spread across different sites or even different continents.

In spite of these complications, developers often overlook the availability of sophisticated and easy-to-use development tools when selecting of a processor for a real-time system. Such tools can have a large impact on meeting time-to-market requirements. A development system that offers its users an expandable array of tools working seamlessly together empowers distributed development teams to focus on innovation, product differentiation, and time-to-market.

2 Development Flow

The Code Composer Studio IDE, an integral component of TI’s software strategy, includes the features necessary to take you through each step of the application development flow. All of these features are provided in an integrated product allowing developers to focus their energy on innovation.

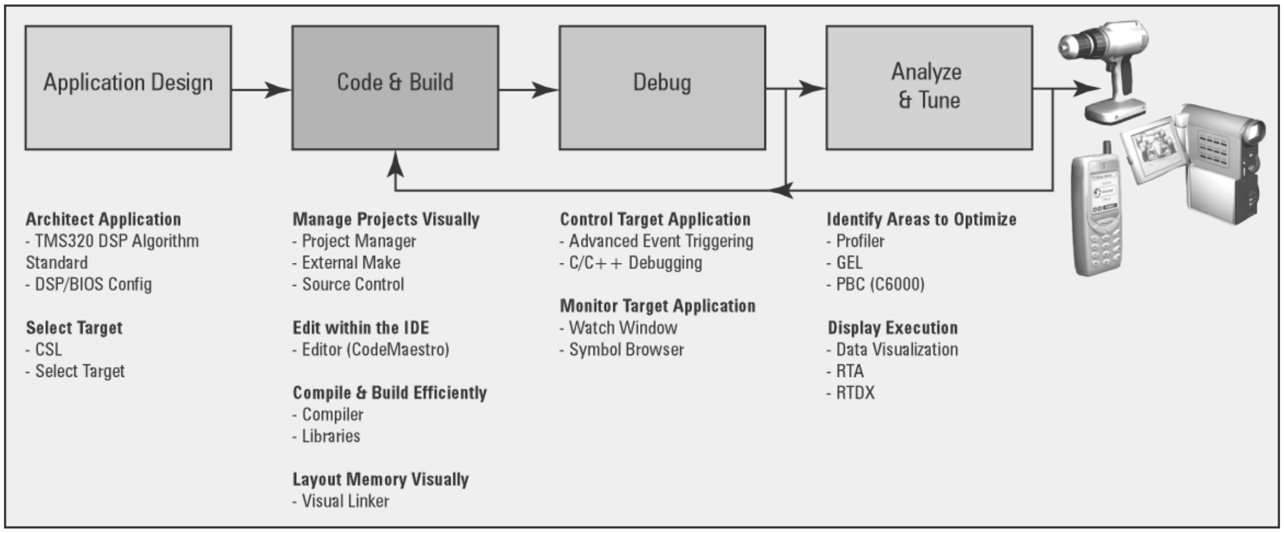


Figure 1. Development Flow

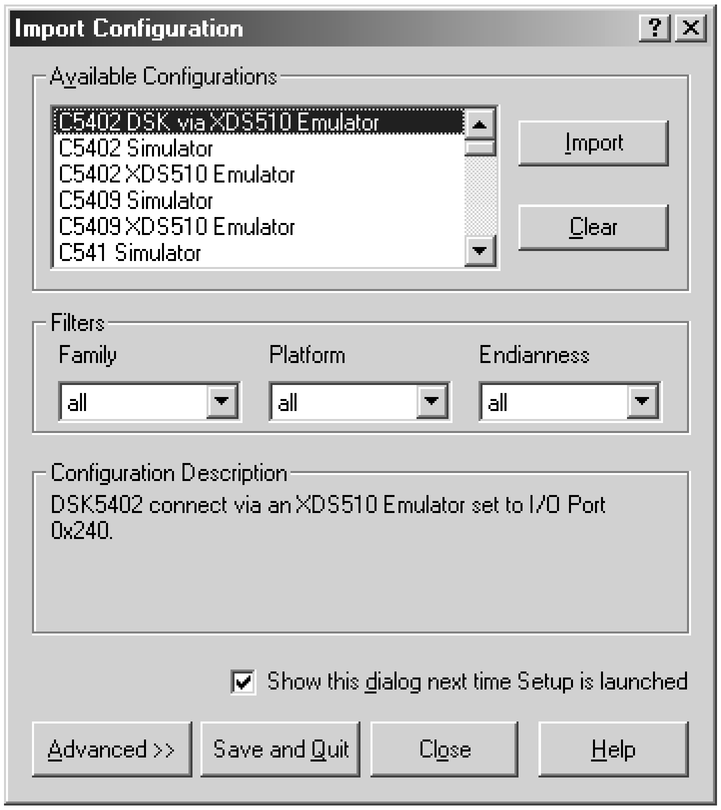
3 Application Design

3.1 Code Composer Studio Setup

Code Composer Studio Setup is a utility that is used to define the target board or simulator you will use with the Code Composer Studio IDE. This information is called the system configuration and consists of a device driver that handles communication with the target plus other information and files that describe the characteristics of your target, such as the default memory map. The Code Composer Studio IDE needs this information to establish communication with your target system and to determine which tools are applicable for the given target.

With the exception of a DSP Starter Kit (DSK), which comes automatically configured for the DSK board, Code Composer Studio IDE will be configured for a simulator by default. You may wish to change the system configuration to match your environment prior to launching Code Composer Studio IDE.

Supplied Configurations. A number of typical configurations are supplied in the form of configuration files (*.ccs). All of these configurations are listed in the Import Configuration Dialog (Figure 2).



C5402 DSK via XDS510 Emulator
C5402 Simulator
C5402 XDS510 Emulator
C5409 Simulator
C5409 XDS510 Emulator
C541 Simulator

Family: all
Platform: all
Endianness: all

DSK5402 connect via an XDS510 Emulator set to I/O Port 0x240.

[✓] Show this dialog next time Setup is launched

Figure 2. Import Configuration Dialog Box

This dialog is shown when you first launch Code Composer Studio Setup, and can also be accessed by selected “Import” from the “File” menu or by clicking on the “Import Configuration File” link at the top of the right-most window pane.

Saving Configurations. Once you have your system configured correctly you can choose to save your configuration to a configuration file. This can then be given to other users who have the same configuration. Saving files is also useful if you are using more than one configuration and often switch between them.

Installing Device Drivers. If you are using an emulator or development board from an eXpressDSP 3rd Party you may need to install the device driver for this target in Code Composer Studio Setup. Some 3rd Parties may install the driver for you automatically.

Using a Custom board.dat File. The board data file contains information that describes the devices on the scan path to the emulator. In Code Composer Studio IDE v2 and later, it is a text file that you can review. In previous versions, the board data file was a binary file. Either format is accepted by Code Composer Studio IDE v2, so that you can specify a board data file from a previous version of Code Composer Studio IDE. If you decide to use a custom board data file instead of the one automatically generated by setup, information in the Processor Configuration tab is not used to create a board data file but is still needed by Code Composer Studio IDE to determine the number and type of control windows to open.

Multiple Processors. Code Composer Studio IDE supports the debugging of multiple processors. When installing a board containing multiple processors the processor configuration tab can be used to specify the number and type of processors on the board.

The Code Composer Studio IDE also provides debug support for multiple processor cores of different ISAs (heterogeneous or co-emulation) connected on the same JTAG scan path. The heterogeneous device driver provided with Code Composer Studio IDE v2 allow you to configure the Code Composer Studio IDE to debug multiple CPUs of different ISAs located on the same JTAG scan path. When the heterogeneous device driver is added to the Code Composer Studio configuration, heterogeneous co-emulation can be performed on any combination of CPU types for which an XDS510 device driver is installed.

GEL Startup Files. Startup files are used to configure the memory map of the debugger for the specific processor that you are using.

3.2 DSP/BIOS

3.2.1 Real-time Operating System

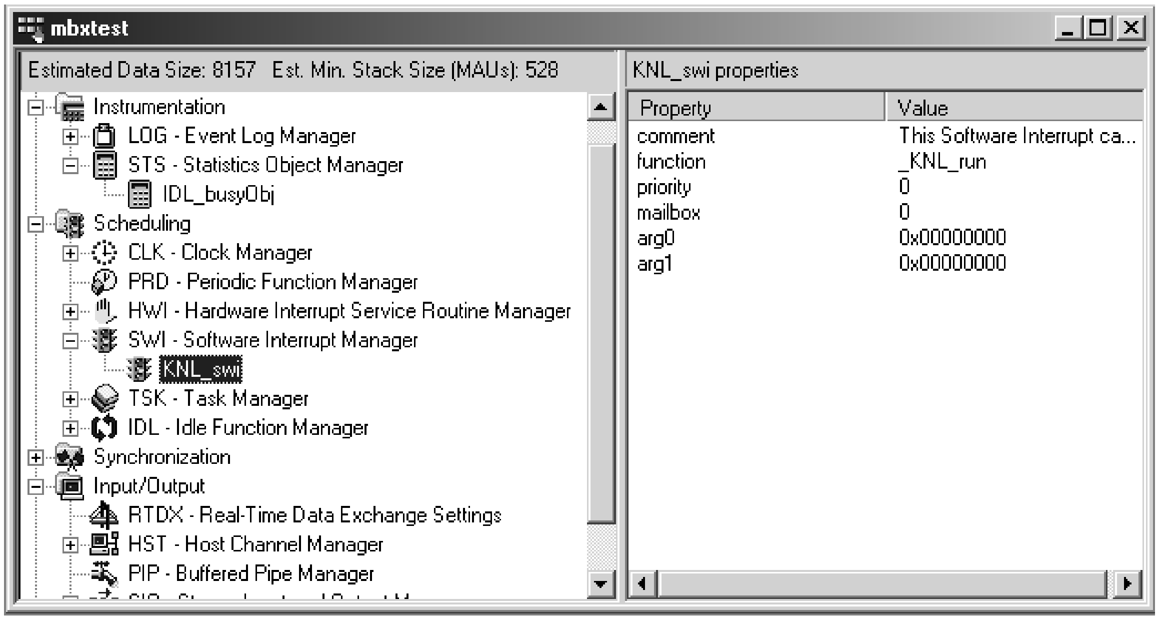
As DSP applications become more complex, it is important to be able to structure them in an efficient, maintainable manner. This requires use of a real-time kernel that enables system functions to be allocated to different threads. The DSP/BIOS kernel provides an efficient set of kernel, real-time analysis, and peripheral configuration services, eliminating the need to develop and maintain custom DSP operating systems. DSP/BIOS is an integral part of the Code Composer Studio IDE. The kernel object browser displays the state of operating system objects, such as tasks and semaphores, and provides data on stack usage and stack overflow/underflow. As a result, developers can more easily debug their application and optimize use the operating system and other system resources.

DSP/BIOS provides four major components that developers can use to get their applications up and running quickly. These consist of a real-time kernel, real-time analysis services, peripheral configuration libraries, and a graphical configuration tool.

DSP/BIOS provides a rich set of deterministic real-time kernel services, callable from C or assembler, which enable developers to create sophisticated applications without compromising real-time deadlines. To provide the fast response required by DSP applications, DSP/BIOS includes Software Interrupts (SWIs) and Periodic Functions (PRDs), in addition to conventional tasks. SWIs are lightweight preemptible threads that share a common stack, resulting in lower memory overhead and faster context switch times. PRDs are time-triggered high-priority threads that can be easily set up to process samples of data arriving at fixed time intervals, simplifying the design of multi-rate systems. To facilitate the design of more complex applications, DSP/BIOS provides interrupt management, I/O mechanisms, and a rich set of inter-task communication services, including semaphores, mailboxes, and queues.

3.2.2 Configuration

The Code Composer Studio IDE provides a graphical host-based configuration tool that makes it easy to for developers to configure all the services provided by DSP/BIOS. All DSP/BIOS objects may be statically configured using this tool. Static configuration shrinks the target memory footprint by removing the code required to perform run-time creation of DSP/BIOS services. It also enables many configuration errors to be detected at build time.


mbxtest

Estimated Data Size: 8157 Est. Min. Stack Size (MAUs): 528

```
Instrumentation
    LOG - Event Log Manager
    STS - Statistics Object Manager
        IDL_busyObj
Scheduling
    CLK - Clock Manager
    PRD - Periodic Function Manager
    HWI - Hardware Interrupt Service Routine Manager
    SWI - Software Interrupt Manager
        KNL_swi
    TSK - Task Manager
    IDL - Idle Function Manager
Synchronization
Input/Output
    RTDX - Real-Time Data Exchange Settings
    HST - Host Channel Manager
    PIP - Buffered Pipe Manager
```

KNL_swi properties

Property	Value
comment	This Software Interrupt ca...
function	_KNL_run
priority	0
mailbox	0
arg0	0x00000000
arg1	0x00000000

Figure 3. Visually Configure Common Run-time Objects

3.2.3 Chip Support Library (CSL)

For each supported processor, DSP/BIOS includes a set of peripheral management functions and macros known as the Chip Support Library (CSL). The CSL supports on-chip peripherals for all TMS320C5000 and TMS320C6000 devices and is fully integrated into the DSP/BIOS Configuration tool. Developer can use the CSL to shorten development time for device driver and initialization code. The CSL supports both graphical and programmatic peripheral configuration and control, eliminating the need to remember individual register flags settings or to painstakingly calculate bit maps.

3.3 TMS320 DSP Algorithm Standard

The TMS320 DSP Algorithm Standard is a set of coding conventions for algorithm writers that reduces time-consuming system integration for anyone trying to put algorithms into their DSP system. This is achieved by defining common programming rules and guidelines with a set of programming interfaces that are consistently used by algorithms across a wide variety of applications.

Content has been created to help both the creators and users of such standard algorithms. This can be found in the TMS320 DSP Algorithm Standard Developer's Kit. For the very first time, this developer's kit has been fully integrated into the Code Composer Studio IDE, with the release of CCStudio 2.0. There is no cost for this developer's kit and no need to download any additional components. The Developer's Kit breaks down into several distinctive areas:

- The TMS320 DSP Algorithm Standard specification
- Application notes for both producers and users of algorithms
- Example code that will build on our common EVM (evaluation module) and DSK (starter kits) hardware platforms
- Tools that aid in the creation of standard header files
- A demonstration that illustrates the simplicity with which algorithms from different suppliers can be easily and quickly integrated into a single application.

By being fully integrated into the CCStudio IDE, the tools that are part of the developer's kit are accessible directly from the tools menu bar at the top of the Code Composer Studio window. A significant amount of online help has been made available to lead the user through, including step-by-step instructions on how to set up and run the demonstration. The demonstration can be started from the CCStudio menu bar.

There are an ever-growing number (350 at time of publication) of algorithms from TI's third party network that are compliant to the algorithm standard and can be easily integrated into applications that are under development using the Code Composer Studio IDE. For further information on the latest list of compliant algorithms visit www.dspvillage.com and select Third Party Compliant solutions.

4 Code and Build

4.1 Source Code Editor

The Code Composer Studio IDE includes fully integrated code editing environment tuned for writing C, C++ and DSP assembly code. The editor provides standard editing features such as: keyword highlighting, printing, cut and paste, drag and drop, etc. The maximum number of lines per file is 2,147,483,648 and the maximum number of characters per line is 3500.

Floating toolbars support advanced operations such as finding the next matching brace and indenting text. The search and replace function makes it easy to change variable names. And to provide users with the most customizable working environment possible, edit windows support either docking to, or floating outside, the parent window in any configuration. Fully integrated with other facilities in the Code Composer Studio IDE like the debugger, the editor allows developers to easily edit code and see both source and disassembly at the same time.

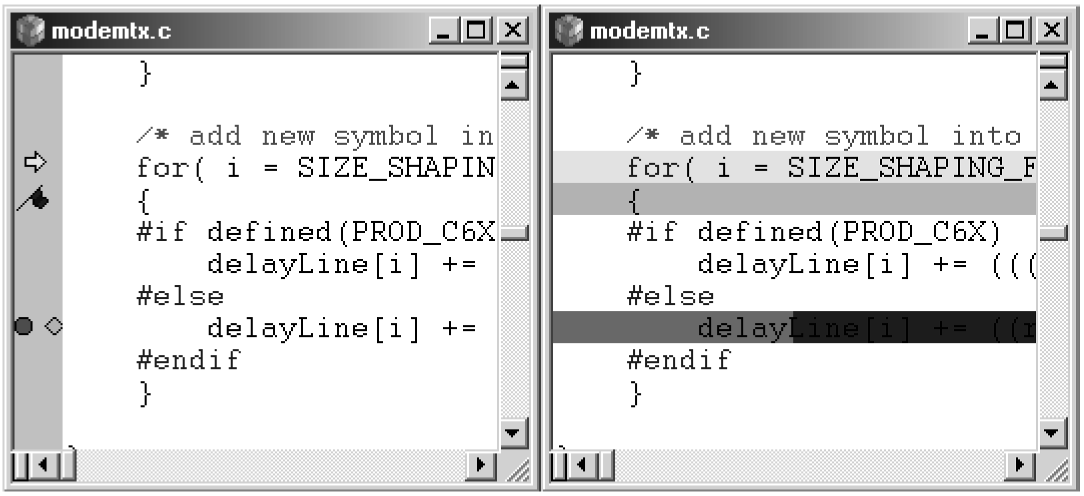
Bookmarks. Use bookmarks to find and maintain key locations within your source files. A bookmark can be set on any line of any source file. Bookmarks are saved with a Code Composer Studio workspace so that they can be recalled at any time.

The Bookmarks dialog box, as well as the bookmarks tab on the project view window, displays the complete list of bookmarks. Use the Bookmarks dialog box to manage all of your bookmarks.

Column Editing. Column editing is used when you want to select, cut, copy, paste, and delete columns of text. Then is very useful when manipulating data structures. To use column editing mode select Edit | Column Editing, or by pressing the keyboard sequence Ctrl + Shift + F8. To toggle between column editing mode and regular mode hold the Alt key and highlight the column. You can then cut, copy, paste, and delete the selected columns as desired.

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).

In the editor, the Selection Margin can be used to set or clear breakpoints. You can also display the line number and the marker points that are set for a given line. It is possible to turn off the selection margin using the Editor Properties tab in the Customize dialog box. The following figure (Figure 4) shows the editor with the Selection Margin on compared with one with it turned off.



```
    }

    /* add new symbol in
    for( i = SIZE_SHAPIN
    {
    #if defined(PROD_C6X
        delayLine[i] +=
    #else
        delayLine[i] +=
    #endif
    }
```

```
    }

    /* add new symbol into
    for( i = SIZE_SHAPING_F
    {
    #if defined(PROD_C6X)
        delayLine[i] += (((
    #else
        delayLine[i] += ((r
    #endif
    }
```

Figure 4. Selection Margin

Mixed Mode. The source code editor supports mixed mode viewing of source. This display mode interleaves the source code with the disassembled instructions that are on the target processor. This mode is useful in determine the actual instructions that are being executed.

A program compiled with “Symbolic Debug” information has a limitation of 65535 lines. When viewing a source file in Mixed Mode, if the file contains more than 65535 lines, the disassembly associated with lines above 65535 is displayed incorrectly. For example, the disassembly for line 65536 is displayed under line 1 of the source code. To work around this problem, compile your program using “Dwarf Debug” information.

Cursor Mode. The editor provides two modes of selecting text and inserting spaces, Stream mode and Virtual Whitespace mode. In Stream mode, the editor provides text selection and insertion similar to most word processors. By default, Stream mode is selected. In Virtual Whitespace mode, the mouse or the arrow keys can be used to move the cursor to any location within the document window. When you begin typing, any whitespace between the end of line and the cursor is automatically filled with spaces. Similarly, when selecting text, any areas beyond the end of line are automatically filled with spaces.

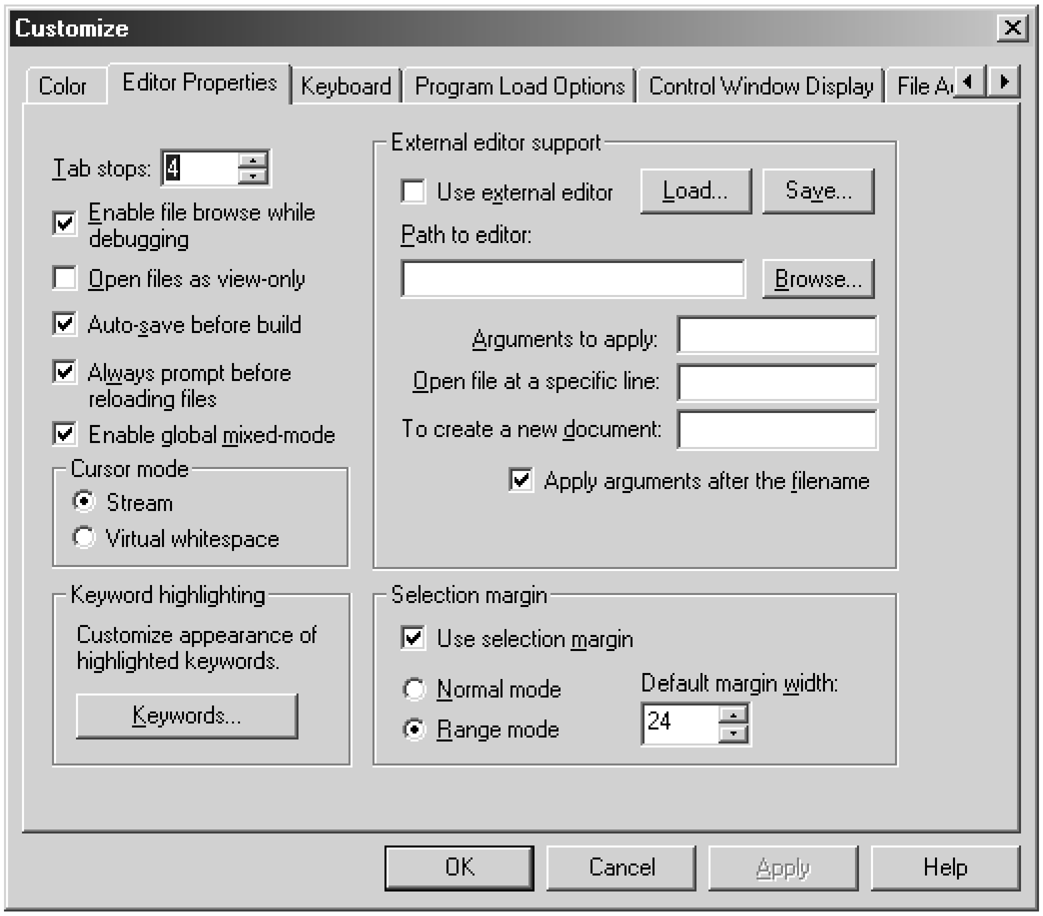
Custom Keywords. The integrated editor features keyword highlighting. Keywords, comments, strings, assembler directives, and GEL commands are highlighted in different colors. Keyword highlighting is available for C, C++, Assembly, and GEL files. When a source file is opened or saved, the editor recognizes the file extension (.c, .cpp, .asm, .gel) and the appropriate keyword highlighting is automatically applied. In addition, new sets of keywords can be created, or the default keyword sets can be customized and saved in keyword files (*.kwd).

Keyword highlighting can be customized according to the file type and the target processor. Keyword files are simple ASCII text files containing a set of keywords separated by carriage returns. You can create a custom keyword file from scratch by using any text editor, or you may make a copy of a default keyword file and then edit it as desired. Four default keyword files are provided. If you installed the Code Composer Studio product in c:\ti, the default keyword files can be found c:\ti\cc\bin\Editor\Keywords\Base. Custom keyword files should be created in c:\ti\cc\bin\Editor\Keywords\Custom. You can add to the list of file extensions that are highlighted.

The Code Composer Studio file keyword.ini stores information about the installed keyword files. The keyword.ini file resides in the Editor folder. If CCStudio cannot find keyword.ini, it will prompt you to enable the default keyword highlighting feature.

4.1.1 External Editor Support

It is possible to use an external source code editor in conjunction with the Code Composer Studio IDE. To configure an external editor you need to specify a number of command line parameters that the editor uses to perform common tasks such as opening a file. The commands can be specified through the Editors Properties tab in the customize dialog. Provided with the Code Composer Studio IDE installation are a number of configuration files that contain these parameters for a number of popular editors. These can be selected through the load button and then look in the external editors directory. Once you have configured an editor you can save the settings to a file to be shared with other members of your team.



Tab stops: 4

[✓] Enable file browse while debugging

[ ] Open files as view-only

[✓] Auto-save before build

[✓] Always prompt before reloading files

[✓] Enable global mixed-mode

Cursor mode: [● Stream] ○ Virtual whitespace

Keyword highlighting: Customize appearance of highlighted keywords.

External editor support

[ ] Use external editor

Path to editor:

Arguments to apply:

Open file at a specific line:

To create a new document:

[✓] Apply arguments after the filename

Selection margin

[✓] Use selection margin

○ Normal mode [● Range mode]

Default margin width: 24

Figure 5. External Editor Settings

4.2 CodeMaestro Technology

Enable advanced editor features. Enable or disable all the options listed in this dialog. Changing this setting causes all open document windows to be saved, closed, and then reopened.

Suggest Word: Automatically suggest words. The editor automatically tries to recognize the word you are typing. When enough characters have been typed to enable the editor to uniquely identify a word, the word is displayed in a pop-up window. To insert the suggested word, press the Tab key.

Complete words. Type the first couple of characters of a word, then press the Tab key to display a list of suggested word completions. To select a word from the list, type enough characters to highlight the desired word.

Automatically display parameter info. When typing a function call, parameter information is automatically displayed.

Type the function name or locate the cursor immediately after a function name. Now type the open parenthesis to begin the parameter list. The editor displays the function declaration in a popup window that appears below the cursor. The first parameter in the list is highlighted. Typing a comma causes the next parameter to be highlighted.

For overloaded functions, the function declarations of all overloaded functions are available. To move forward or backward through the list of function declarations, press Ctrl + Page Up or Ctrl + Page Down.

To close the popup window, press the Escape key. To reopen the popup window, press Ctrl + Shift + Space. The popup window also closes when you complete the parameter list and type the close parenthesis.

Automatically list members. A member function or variable is specified using the syntax object.member. A list of the members available for an object is displayed after you type the period that separates the object name and the member name. To select a member from the list, type enough characters to highlight the desired member.

Automatically correct case. When selected, the editor tries to identify the keyword that you are typing. After typing the space following the end of the keyword, the editor automatically converts the word to the proper case.

4.3 Project Manager

The Project Management system has been broadened and improved to include multi project/multi configuration support and source code control, as well as the ability to interface to external build processes. The new Project Manager can support projects containing thousands of source files. These features were designed with large distributed development teams in mind.

The project manager organizes your files into folders for source, library, configuration, include and generated files. This allows you to easily navigate through your source files. The project manager automatically scans dependencies removing the need to add dependent files to the project. These files are shown in the include folder in the project view window.

Project File. The project file itself is a text file that can be edited manually. This is useful for batch environments or for generating the project file via a script. The project file stores all of the build options for each configuration in the project.

4.3.1 Project Configurations

The Code Composer Studio Project Manager now supports multiple build configurations. Multiple build configurations allow, for example, a Debug Configuration and a Release Configuration of build options. Separate configurations for Debug and Release are helpful because they allow for two independent builds without making any changes to the project options. The compile/link options separate and independent folders are created to hold all the output files generated during the build process. The names of the folders match the configuration name so it is easy to track where the output files for a particular configuration are. Any build configuration that you want to capture and save can be stored and later retrieved for bit exact rebuilds.

4.3.2 Multiple Project Support

It is quite typical for a developer to have more than one project going at once. They may have some maintenance work to do on an old project, another project that represents the next product release and perhaps some exploratory work as well. The Code Composer Studio Project Manager allows you to have multiple projects open simultaneously and easily switch between them. With larger applications it is likely that the work will be divided up into a series of libraries each represented by a different project file. Code Composer Studio IDE allows you to easily bring these projects together.

4.3.3 Source Control

Code Composer Studio IDE allows a developer the capability to interface a preferred source code control software application directly through the Project Manager or pull-down menus. The Code Composer Studio IDE supports any source control provider that correctly implements the Microsoft SCC Interface. Some of the more popular source control systems that are compatible with CCStudio are Rational's ClearCase, Microsoft's SourceSafe, Starbase StarTeam PVCS or MKS. Once CCStudio is interfaced with the source code control application, the project will display visually which files are checked in or out. Source code control is essential for software development projects of all sizes. It helps with revision control, branching and merging of changes, and backups.

The project view window will list all the source files that are in a particular project in the source folder. Files that are 'checked out' to the source control system will have green checks to the left of the name. If a file is checked out, it can be opened in an Editor window and modified accordingly. Files that are 'checked in' to the source code control system will be indicated by red checks to the left of the name. These 'checked in' files can be opened for viewing in an Editor window but they are read only so changes cannot be saved.

Once a source code control provider is selected, that provider's tools are used whenever any of the source code control commands are invoked within the Code Composer Studio IDE. The source code control features are limited by the capabilities of the provider.

4.3.4 Building Libraries

Code Composer Studio IDE supports the creation of archives. When creating a new project you can specify if the project is an executable or a library. When building a library project all the build options are selectable as with any other project except the linker. Since there is no executable output the linker is not needed and this is replaced with the options for the archiver.

4.3.5 Custom Build Steps

Custom build steps are extremely helpful at automating steps that are done before and/or after the compiling/linking phase. An example of a post-build step would be converting the coff output file to hex format file. An example of a step that could be automated either pre- or post-build is copying files from one location to another.

If file-specific custom build options are required they can be specified here in the file specific build options dialog box. You may want to exclude a particular file from a build while still keeping it in the project. These options are used to override the standard build tools and replace them with user specific tools. Typically this is used in conjunction with external makefiles.

Exclude file from build. The selected file is excluded from the build process. By default, this option is turned off.

Use custom build step. Activate the Custom Build Setting fields. Choosing this option enables you to specify custom build commands. By default, this option is turned off.

Custom Build Setting. After selecting "Use custom build step", you can specify custom build commands for the selected file.

Build Command. Enter the command line to be executed to complete a build.

Outputs. Enter a path and filename for the executable file and any intermediate files generated by the build command.

Clean Command. Enter the command line to be executed to remove any files generated by the build command.

Important: If any custom build step involves launching Texas Instruments code generation tools, you must execute the batch file DosRun.bat to set up the necessary PATH and environment variables (C_DIR, A_DIR…). It is recommended that you execute DosRun.bat as the first step of the initial build steps. If Code Composer Studio IDE is installed in the default location, the batch file is located in the directory the main installation directory, which is usually c:\ti.

4.3.6 Using External Makefiles

This feature of the Project Manager allows the developer to use a makefile (*.mak) created for a legacy product development. If for instance you had an existing build process outside of Code Composer Studio IDE and now you wanted to build within Code Composer Studio IDE, you would use the “Use External Makefile” build option. The existing makefile will be called from within Code Composer Studio IDE, giving the effect of a fully integrated build environment. It is necessary to specify the commands used to launch the make process and how source files are defined in the makefile so that the makefile can be parsed and the source files displayed in the source folder of the Project View. The parsing is done using regular expressions.

A Code Composer Studio project is created that contains the external makefile. Now that a project is associated with the makefile, the project and its contents can be displayed in the Project View window and the Project→Build and Project→Rebuild All commands can be used to build the project.

4.3.7 Exporting Makefiles

A Code Composer Studio project can be exported to a standard makefile, saving the user the time of manually recreating the extensive build steps followed by Code Composer Studio IDE. The makefile is compatible with any make utility derived from the standard make utility. Code development projects can be developed using the Code Composer Studio IDE, while final builds are done in some other environment., even be on another operating system. By using the Export to a makefile utility a makefile is created for use in this environment. This may be necessary if your build environment is on a UNIX platform.

4.3.8 Command-Line Build Utility (timake.exe)

Code Composer Studio IDE provides an external build utility that allows projects (*.pjt) to be built outside of the IDE. This allows projects to be integrated into larger system builds. These system builds, for example, could include a host processor. Now the DSP and Host processor code bases can be built from one master makefile or batch file. The master makefile or batch file will call the timake utility to build the DSP portion of the system. The timake.exe utility takes a *.pjt file and other options and generates an output file specified by the options set in the *.pjt file.

4.4 Code Generation Tools

In the past, developing high performance DSP code has required to developer to optimize assembly code by hand and to have an intimate knowledge of the particular DSP architecture. Because time-to-market is becoming increasingly important, while the time and skill to optimally code a DSP are increasingly hard to find, there is a need for a more robust code development environment. The Code Composer Studio compile tools address this need by shifting the burden of optimization from hand-coded assembly to the C Compiler. With these tools it is possible to exploit the high performance of TI's DSP platforms without ever writing hand-coded assembly.

4.4.1 C6000 Code Generation Tools

For the embedded software developer, TI' s C6000 Compile Tools – co-developed with the DSP architecture – offer best in class performance due to industry leading global view analysis and architecture specific optimizations including interactive profiling, tuning and feedback.

One of the major areas of focus for the C6000 compiler has been VLIW architecture specific optimizations. Since the release 1.0 of the C6000 tools in Feb 1997, TI has continued to drive C performance improvements on key DSP MIPS intensive code. Software pipelining, data path partitioning, inner and nested loop optimization, unrolling, and predication form the basis of the C6000 specific enhancements included.

In addition to this core set of optimizations, TI offers something unique in the industry called compiler feedback, telling the developer exactly how the compiler did and what potential areas could be improved. This, used in conjunction with the compiler tutorial feedback solutions, can be used to provide the developer valuable insight when tuning C code.

To fully understand what makes the C6000 Compile tools best in industry be sure to check out the C6000 Compiler Optimization Tutorial, Assembly Optimizer and Profile Based Compilation as well.

4.4.2 C5000 Code Generation Tools

Code Composer Studio IDE v2 offers a 50% increase in C/C++ compiler performance on the new TMS320C55x™ devices. In addition, this release extends upon industry leading control code size efficiency. Along with the C/C++ compiler, assembly language tools provide allow for a fully compatible upgrade path from TMS320C54x™ assembly.

TMS320C55x and TMS320C54x are trademarks of Texas Instruments.

5 Debug

5.1 Debugger

The Code Composer Studio debugger helps you find and fix errors in your real-time embedded software programs. Debugger commands enable you to control program execution. Debugger windows and dialogs allow you to view source code and track the values of program variables in memory and registers. Breakpoints enable you to stop execution at a specified location and examine the current state of the program.

Memory Window. The Memory window allows you to view the contents of memory starting at a specified address. Options enable you to format the Memory window display. You can also edit the contents of a selected memory location.

Registers Window. The Register windows enable you to view and edit the contents of the CPU registers and the Peripheral Registers.

Disassembly Window. The Disassembly window displays disassembled instructions and symbolic information needed for debugging. Disassembly reverses the assembly process and allows the contents of memory to be displayed as assembly language code. Symbolic information consists of symbols and strings of alphanumeric characters that represent addresses or values on the target.

Call Stack. Use the Call Stack window to examine the function calls that led to the current location in the program that you are debugging. The call stack only works with C programs. Calling functions are determined by walking through the linked list of frame pointers on the runtime stack. Your program must have a stack section and a main function; otherwise, the call stack displays following the message: C source is not available.

Symbol Browser. The Symbol Browser displays all of the associated files, functions, global variables, types, and labels of a loaded COFF output file (*.out). From the Symbol Browser you can open the source code for a file or function in the text editor. You can also use the Symbol Browser to create a profile area. When creating a C++ application the Symbol Browser will function as a C++ class browser.

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. The Watch Locals tab of the Watch window provides a special type of Watch window. In this window, the debugger automatically displays the values of variables that are local to the currently executing function. You can change the format used to display the value of a watch variable or expression. The default display format is based on the type of the variable.

Command Window. The Command Window enables you to specify commands to the Code Composer Studio 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.

5.1.1 *Probe Points*

A Probe Point™ can be set at any point in your algorithm (similar to the way a breakpoint is set). When the execution of the program reaches a Probe Point, the connected object (whether it is a file, graph, or Memory window) is updated. Once the connected object is updated, execution continues. When Probe Points are set, you can enable or disable them just like breakpoints.

When a window is created, by default, it is updated at every breakpoint. However, you can change this so the window is updated only when the program reaches the connected Probe Point.

File I/O. The debugger allows you to stream, or transfer, data to or from the actual/simulated target from a PC file. This is a great way to simulate your code using known sample values. The File I/O feature uses Probe Points , which allow you to extract/inject samples or take a snapshot of memory locations at a point you define (Probe Point). If you set a Probe Point at a specific point in your code and then connect a file to it, you can implement file I/O functionality. File I/O coupled with Probe Points are two excellent tools that can help developers automate test cases against a know set of resultant values.

5.1.2 *Parallel Debug Manager*

The Parallel Debug Manager (PDM) allows you to synchronize multiple processors. If you have several processors and a device driver that supports them, the PDM is enabled when you start Code Composer Studio IDE. From the PDM menu bar, you can open individual parent windows to control each processor or you can broadcast commands to a group of processors.

Synchronized Commands. All commands in the Parallel Debug Manager (PDM) are broadcast to all target processors in the current group. If the device driver supports synchronous operation these commands are synchronized to start at the same time on each processor. The following commands are supported: Locked Step, StepOver, StepOut, Run, Halt and Animate.

5.2 Real-Time Data Exchange (RTDX)

The data visualization tools included with Code Composer Studio IDE (described in detail in section 6.1) are excellent for proving the correctness of DSP algorithms. Once algorithms are integrated into applications, the real-time behavior of the system must be observed. Real-Time Data Exchange, or RTDX, provides significant benefits over alternative methods of system visualization. Until recently, developers were forced to stop their application with a breakpoint to exchange data ”snapshots” with the host computer, in a technique that’s called ”stop-mode debugging”. This intrusive method of debugging may yield misleading information because the isolated snapshot of a halted high-speed application does not present an accurate view of the system’s real-world operation.

RTDX is a technology that was invented by Texas Instruments, just as its foundation, JTAG emulation was. RTDX gives designers the industry’s first DSP system that provides real-time, continuous visibility into the way target applications operate in the real world. RTDX allows developers to transfer data between the host computer and DSP devices without stopping their target application.

TMS320C6000, C6000, TMS320C5000, C5000, Code Composer Studio, CodeMaestro, DSP/BIOS and RTDX are trademarks of Texas Instruments.

This important breakthrough shortens development time by giving developers a much more realistic representation of the way their systems operate. Just as modern medical diagnostic equipment provides a real-time, ongoing analysis of the way a patient's body is functioning, RTDX allows designers to continually monitor their systems and gain real-time insight into their running applications.

RTDX runs asynchronously, in real-time, between the target and the host, and is also multiplexed. RTDX can transfer data bi-directionally, and thus allows the developer to either access data at a specific point in their application and visualize that data as it changes in real time as the application executes, or to simulate data input down to the DSP from (for example) a sensor.

RTDX is implemented with a small monitor (less than 2 Kilobytes) that is included as a part of DSP/BIOS. The target application makes calls to RTDX functions as needed to log the data back to the host over the JTAG connection, and the data can then be logged in either continuous or snapshot modes.

When the RTDX data is received on the host, it is available from Code Composer Studio IDE through an industry standard Microsoft ActiveX API that allows separate applications to communicate with each other. This is a key capability, as it allows RTDX data to be streamed into any other ActiveX-compliant application, including common desktop data visualization software such as Microsoft Excel, Visual Basic, Visual C++, and Visual J++, data acquisition packages such as National Instruments LabVIEW, and signal processing analysis systems such as MATLAB from The Math Works. Developers can also easily create their own visualization applications, since they know best how to visualize or analyze the application data. To make things run right out of the box, there is even a general-purpose display that is included with the product to help developers view their data in a standard format as well.

When displaying RTDX data, host applications can read either live or saved data, that is, data that is being received in real-time from the executing application, or data that has already been recorded in a log file, and is being ”played back”. Multiple host applications can even be run simultaneously, all reading or writing RTDX data at the same time.

With Code Composer Studio IDE v2 it is now possible to run RTDX with the instruction simulator, allowing RTDX to be used before target hardware is available. This also makes it possible to use DSP/BIOS Real-Time Analysis (RTA) with only the simulator, as RTDX is the data transfer mechanism used for Real-Time Analysis. Another new RTDX capability in Code Composer Studio IDE v2 is heterogeneous multiprocessor support, where RTDX can run on multiple targets simultaneously. This support is available for the C6000, C5000, and TMS470 ISA families. Heterogeneous multiprocessor RTDX can be used with multiple processors on either the same or different scan paths. One advanced use of heterogeneous multiprocessor RTDX is to have multiple processors communicating with each other via RTDX, where there can theoretically be any number of processors communicating with each other.

The following tools are used to manage RTDX:

Diagnostics Tool: This is the main RTDX Diagnostics Control window of the RTDX Configuration Control. You can use the RTDX Diagnostics Control to verify that RTDX is working correctly on your system. The diagnostic tests test the basic functionality testing the Target-To-Host transmission and the Host-To-Target transmission.

Configuration Tool: This is the main RTDX Configuration Control window. It allows you to view the current RTDX configuration settings, enable/disable RTDX and access the RTDX Configuration Control Properties window for configuring the RTDX setup.

Channel Viewer: This is the main RTDX Channel Viewer Control window of the RTDX Configuration Control. The RTDX Channel Viewer Control is an ActiveX control that automatically detects target-declared channels and adds them to the viewable list. The RTDX Channel Viewer Control also allows you to remove or re-add target-declared channels to the viewable list when desired, and enable or disable channels once they have been added to the list.

5.3 Advanced Event Triggering (AET)

Advanced Event Triggering is designed to make hardware analysis much easier that in the past. AET is only available on new hardware targets that have the appropriate analysis interface.

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

Event Sequencer. Event Sequencer allows you to look for conditions to occur in your target program and cause actions to occur when those conditions are detected. While the CPU is halted, you define the conditions and actions, and then run your target program. The sequence program then looks for the condition and performs the action you requested.

6 Analyze and Tune

6.1 Data Visualization

Code Composer Studio IDE incorporates an advanced signal analysis interface that enables developers to monitor signal data critically and thoroughly. The new features are useful in developing applications for communications, wireless, and image processing, as well as for general DSP applications.

Time/frequency. You can use a time/frequency graph to view signals in either the time or frequency domain. For frequency domain analysis, the display buffer is run through an FFT routine to give a frequency domain analysis of the data. Frequency graphs include FFT Magnitude, FFT Waterfall, Complex FFT, and FFT Magnitude and Phase. In time domain analysis, no preprocessing is done on the display data before it is graphed. Time domain graphs can be either single or dual time.

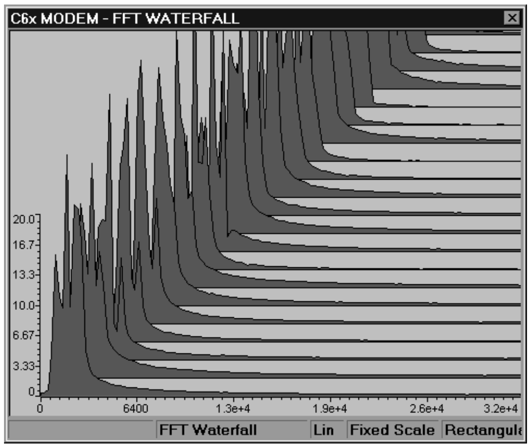


Figure 6. FFT Waterfall

Constellation Plot. You can use a constellation graph to measure how effectively the information is extracted from the input signal. The input signal is separated into two components and the resulting data is plotted using the Cartesian coordinate system in time, by plotting one signal versus the other (Y source versus X source, where Y is plotted on the Y axis and X on the X axis).

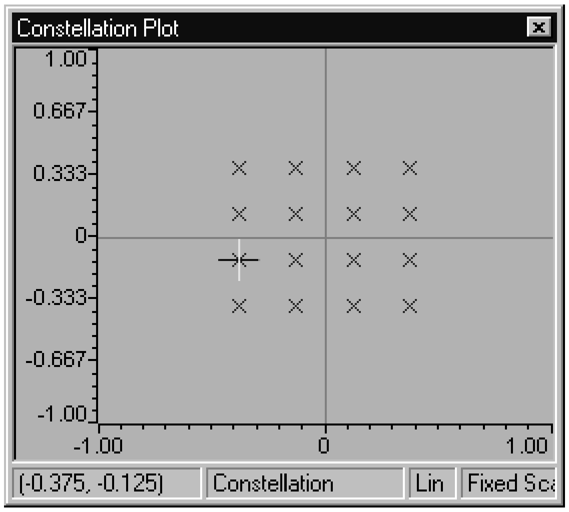


Figure 7. Constellation Plot

Eye Diagram. You can use an eye diagram to qualitatively examine signal fidelity. Incoming signals are continuously superimposed upon each other within a specified display range and are displayed in an eye shape. The signal's period is shown over time by plotting the signal serially and wrapping it back when 0 crossings are detected. These are reference points at which a signal (specified by the data source) can wrap back to the beginning of the window frame.

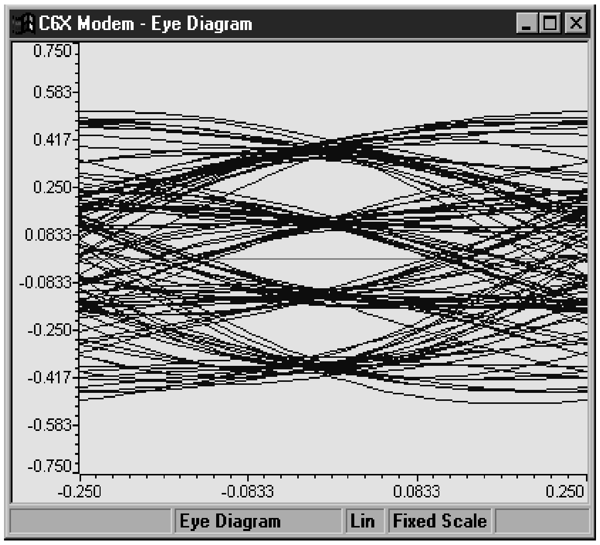


Figure 8. Eye Diagram

Image Display. You can use an image graph to test image-processing algorithms. Image data is displayed based on RGB and YUV data streams.

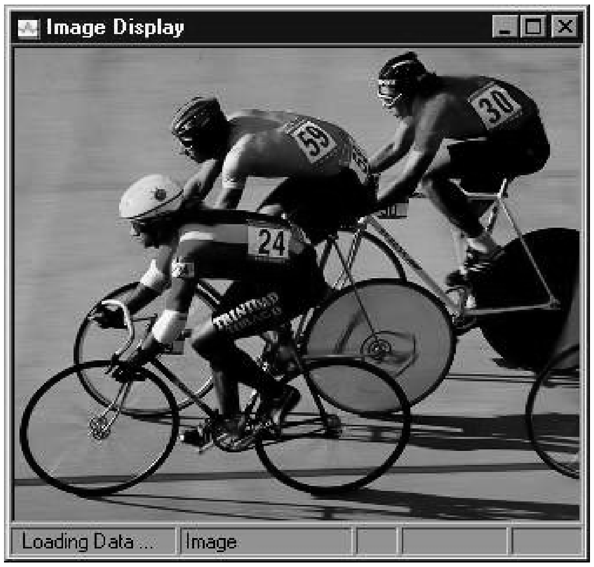


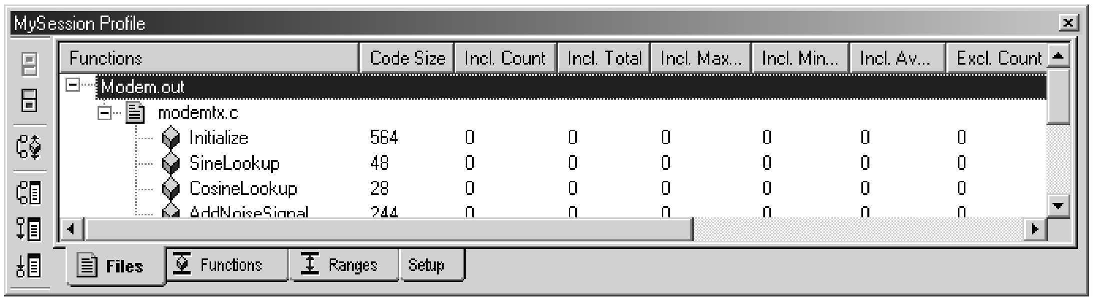
Figure 9. Image Display

6.2 Interactive Profiler

Performance is a key issue for embedded systems developers. As programs grow in size and complexity it becomes more difficult for developers to isolate the subtle problems that cause poor performance.

Profiling helps reduce the time it takes to identify and eliminate performance bottlenecks. The profiler analyzes program execution and shows 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.



Functions	Code Size	Incl. Count	Incl. Total	Incl. Max...	Incl. Min...	Incl. Av...	Excl. Count
Modem.out							
modemtx.c							
Initialize	564	0	0	0	0	0	0
SineLookup	48	0	0	0	0	0	0
CosineLookup	28	0	0	0	0	0	0
AddNoiseSignal	244	0	0	0	0	0	0

Figure 10. Profiler

Function Profiling. The profiler is very useful for profile C or C++ functions. For Assembly functions are not currently supported and should be treated as ranges. There are a number of ways to add a function to the profiler. The first is to highlight the function in your source window and drag it into the profiler. Alternatively you can use a dialog box to enter in the file name and lines that you wish to profile. If you are using the symbol browser you can also drag functions from it into the profiler. Figure 10 shows a number of functions being profiled.

Range Profiling. The second type of profile area supported is what we refer to as a range. A range can be a range of source lines as seen on the slide. You can simply select a range in the edit window and drag it into the ranges tab on the profiler or simply right click on the selection and select add to profiling session. If you mistakenly drag the selection into the functions tab it will interpret your action as a desire to profile the function that the selection is located in and add the function to the profiler session. You can also add a range of source lines using the dialog box that was shown earlier. It is also possible to profile a range of disassembly or DSP addresses. You can do this by selecting a range of code in the visible region of the disassembly window and dragging it into the ranges tab on the profiler window. If you want to select a range that will not fit within the visible region within the window (I.e. requires scrolling) then you will need to use the add profile area dialog box.

Create Statistics Report. The new profiler has a feature called “Create Statistics Report”. By clicking on this button located at the bottom left of the profiler toolbar you can create a tab delimited text file. Once you create a report it is opened in the CCStudio window for quick review. However, the simple format of this file is intended for importing into spreadsheet applications.

Start and End Points. When profiling there are often sections of code that you do not want including in your profiling results. This could be due to the fact that a particular range of code is already optimized, or it may be startup or initialization code. The Code Composer Studio profiler supports an advance profiling feature call start and end points. These points are used to turn the profile on or off at a specific point in the code.

For example to profile a function excluding a specific loop within the function from the profiling results you would add the function to the profiler and then set an end point at the beginning of the loop and then a start point at the end of the loop. This would ensure that the cycles for that loop are excluded from my results. The setup tab on the profiler window displays all of the start and end points. You can place these points by dragging a line into this tab or by using the “Create Setup Start Point” or “Create Setup End Point” buttons located on the profiler toolbar.

6.3 Profile-Based Compilation (PBC)

As demands for real time processing rise, DSP architectures continue to push the performance envelope by increasing the amount of parallelism available to the programmer. At the same time applications are becoming more complex and larger in scope and code size. As TI’s C6000 platform is the performance leader in the industry it’s only fitting that TI is the first to offer a new C6000 tool, Profile Based Compilation (PBC). PBC is aimed at helping the embedded software developer make the right tradeoffs in code size and performance automatically. This novel interface actually builds and profiles multiple build option sets and then allows the user to automatically select along a two dimensional graph, the desired performance and code size needed for the application. The result: performance and code size that meet the developer’s system requirements.

6.3.1 PBC Tutorial and Demonstration

A self-paced demo/tutorial begins automatically the first time you run PBC. Included with the demo/tutorial is a code example ready to run. Simply follow the wizard as it walks you through six easy steps with the click of a mouse to make the right code size vs. performance tradeoff.

Once you select the performance and code size you want, the solution window displays every function's cycle count, code size, and option set in a window that can be sorted by each column. Once you are satisfied with the selected solution, simply click build and each function is built with the appropriate compiler option sets.

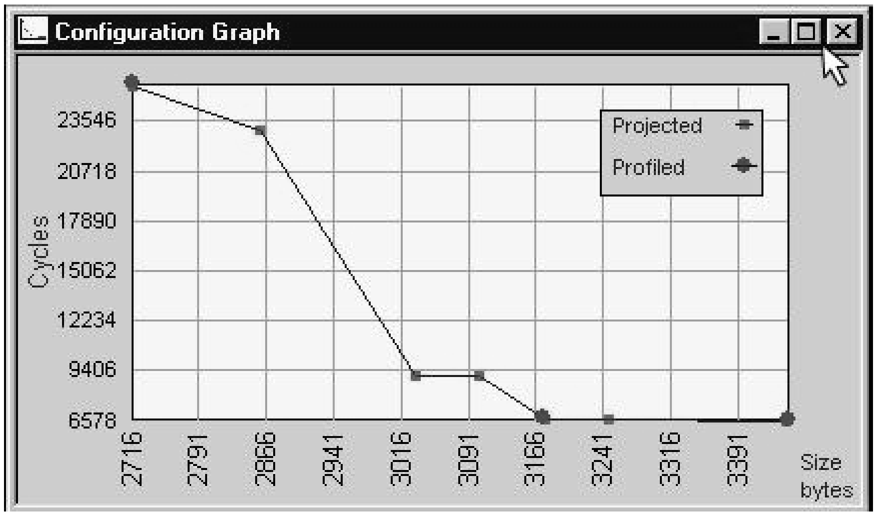


Figure 11. Trade-off: Code Size vs. Speed

6.4 Visual Linker

Linking is a process that is largely unnoticed or ignored in native development where the use of virtual memory makes it a non-issue. However, linking is a critical part of the application development process for real-time DSP applications. In embedded systems matching the parts of a DSP application with the correct types, speeds, and sizes of memory (on-chip, SRAM, DRAM, SARAM, DARAM, PROM, FLASH, etc.) means the difference between meeting and missing real-time deadlines. The criticality of program and data placement to performance, the increasing size of applications, and the increasing variety of types of memory has driven the need for a more graphical, interactive linking environment for embedded system developers.

The new Visual Linker for the C5000 and C6000 DSP platforms dramatically simplifies linking, reduces time-to-market, decreases application size, and helps boost performance. Utilizing a Windows Explorer like interface, the Visual Linker provides the ability to: drag-and-drop program components into multiple and different memory types and areas; choose from a library of standard memory maps; uncover opportunities for optimization using immediate visual feedback on memory allocation; and reduce application size with fine-grained automatic elimination of unused code and data.

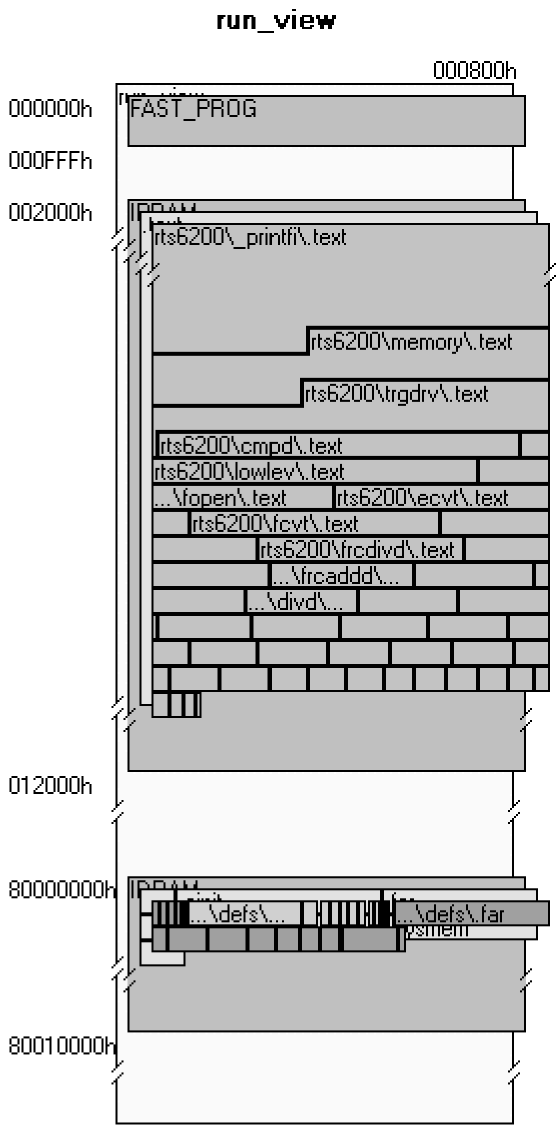


Figure 12. Visually Arranged Code Placement

6.5 Real-Time Analysis

To properly assess the performance of a system, developers must be able to view both the data output and the timing relationships between tasks in the system. In years past, developers had to implement their own instrumentation as they progressed through testing and evaluation. The increasing complexity of today's systems with high throughput, multiple data channels, and multiple tasks, demands a new approach.

Real-time analysis tools are required to act as a visualization aid allowing developers to see the time-related interactions between code sections. This can bring to light subtle real-time problems that might otherwise go undetected. Developers need tools that provide the capability to analyze data acquired during the real-time execution of software – i.e. a software logic analyzer. In order for these tools to function it is necessary for there to exist a real-time link and awareness between the host development environment and the DSP target. By addressing and meeting these needs, Code Composer Studio IDE allows developers to probe, trace and monitor a DSP application while it runs. Even after the program has halted, information already captured through real-time analysis tools can provide invaluable insight into the sequence of events that led up to the current point of execution. As real-time system developers say, 'If a problem can be found then it can be fixed'.

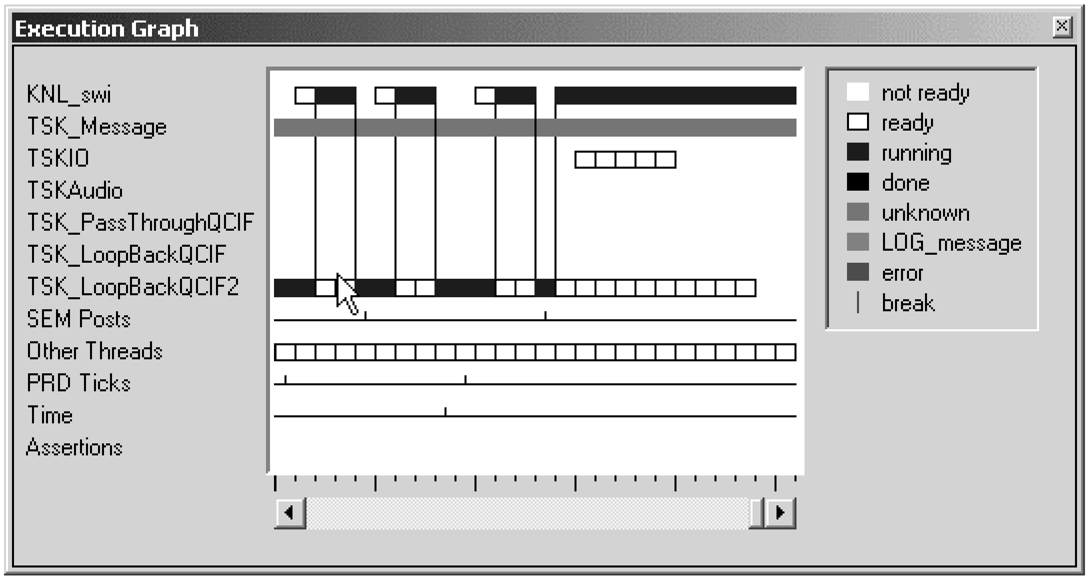


Figure 13. Find and Fix Subtle Real-Time Problems

Debug and analysis is essentially a double-edged sword. Early in development users are concerned with the algorithm returning the correct data values. Later in development, as developers try to achieve maximum integration and functionality, they need to see the subtle timing relationships between threads. The intrinsic diagnostic features of Code Composer Studio IDE's API's on the target give developers low overhead statistics and control. The CPU load graph illustrates the level of processor loading against program execution, enabling developers to determine areas where CPU resources are being stretched or where there is headroom to add more functions.

Real-time analysis tools are used when transitioning from the debug phase to the run-time phase of development. At this time, the tools can show subtle problems arising from time-dependent interaction of program components. These tools are the software counterpart of the ubiquitous hardware logic analyzer. In addition, the real-time logging/analysis capabilities provided by DSP/BIOS and RTDX technology are the foundation paving the way for a new generation of manufacturing and field diagnostic tools that can analyze and troubleshoot in the field.

6.6 General Extension Language (GEL)

The General Extension Language (GEL) is an interpretive language similar to C that lets you create functions to extend Code Composer Studio IDE's usefulness. You create your GEL functions using the GEL grammar and then load them into Code Composer Studio IDE. With GEL, you can access actual/simulated target memory locations and add options to Code Composer Studio IDE's GEL menu. GEL is particularly useful for automated testing. 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.

Keywords are used to add GEL functions to the GEL menu of Code Composer Studio IDE. The menuitem keyword is used to create a new drop-down list of items in the GEL menu. You can then use the keywords hotmenu, dialog, or slider to add new menu items. When you select a user defined menu item (under the GEL menu), a dialog box or slider object appears. The hotmenu keyword adds a GEL function to the GEL menu and when selected, it immediately executes that specific function. The dialog keyword creates a GUI dialog window for parameter entry. Once the appropriate parameters are entered, hit the execute button to call the function. The slider keyword creates an adjustable controller that can vary the value of a single parameter.

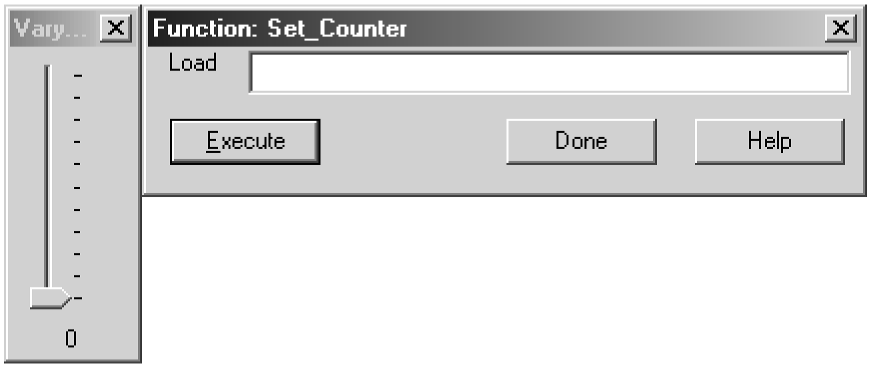


Figure 14. GEL Controls

6.7 Open Plug-In Architecture

Today's highly complex system designs often require hardware and software tools with more capabilities than those previously available. Code Composer Studio IDE's open plug-in architecture allows developers to easily extend the IDE to match their growing needs. By leveraging DSP's most extensive third-party network, designers can select third-party tools to plug into the Code Composer Studio development environment. Not only is this a better alternative to buying separate applications to meet new requirements, it also helps reduce the risk to developers. By allocating fewer resources to developing individual utilities, developers can spend more time testing the end product, resulting in increasing an application's robustness.

7 Summary

Code Composer Studio IDE v2.0 represents the evolution of the DSP development environment. It contains all of the functionality needed by today’s larger, distributed, global project teams. The intelligent IDE can help save valuable development time by making developers more productive enabling them to focus their energies on innovation instead of repetitive tasks and tool development.