Code Composer Studio IDE v3 White Paper

John Stevenson Texas Instruments Incorporated

ABSTRACT

Designed for the Texas Instruments (TI) high performance TMS320C6000™ (C6000™) and 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. Version 3.0 (Tuning Edition) includes tuning tools for optimizing applications.

Key components of the Code Composer Studio IDE include:

- Tuning tools for optimizing applications
- C/C++ Compiler, Assembly Optimizer and Linker (Code Generation Tools)
- Real-Time Operating System (DSP/BIOS™)
- Ability to dynamically connect and disconnect from targets
- Real-Time Data Exchange between host and target (RTDX™)
- Update Advisor
- Instruction Set Simulator
- Advanced Event Triggering
- Data Visualization

The Code Composer Studio IDE Version 3.0 (Tuning Edition) 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 2000 or Windows XP, with Internet Explorer 5.5 or later

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 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.

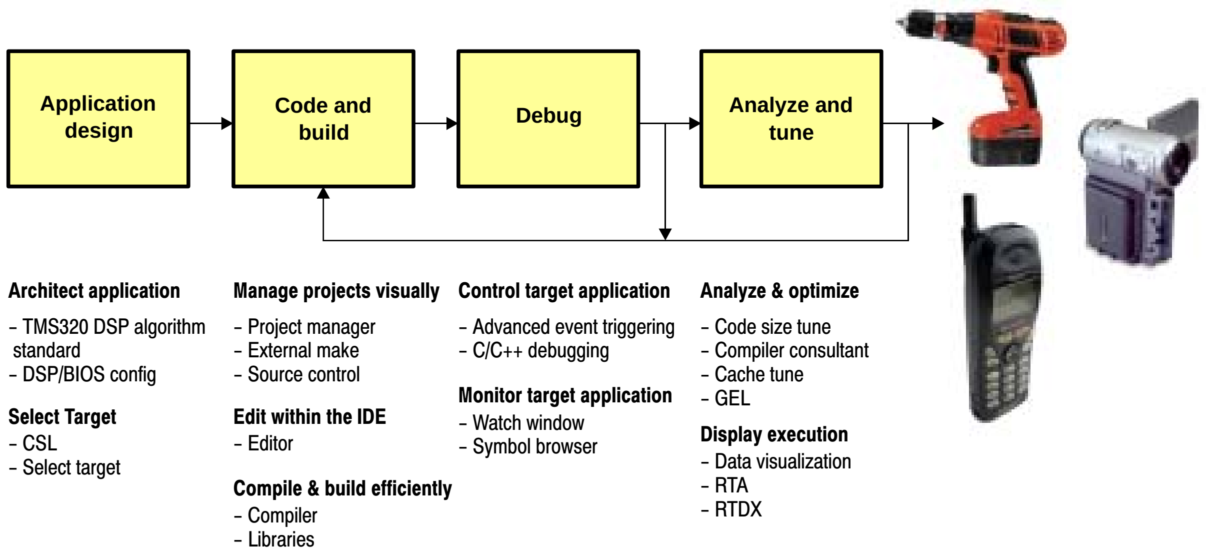


Figure 1. Development Flow

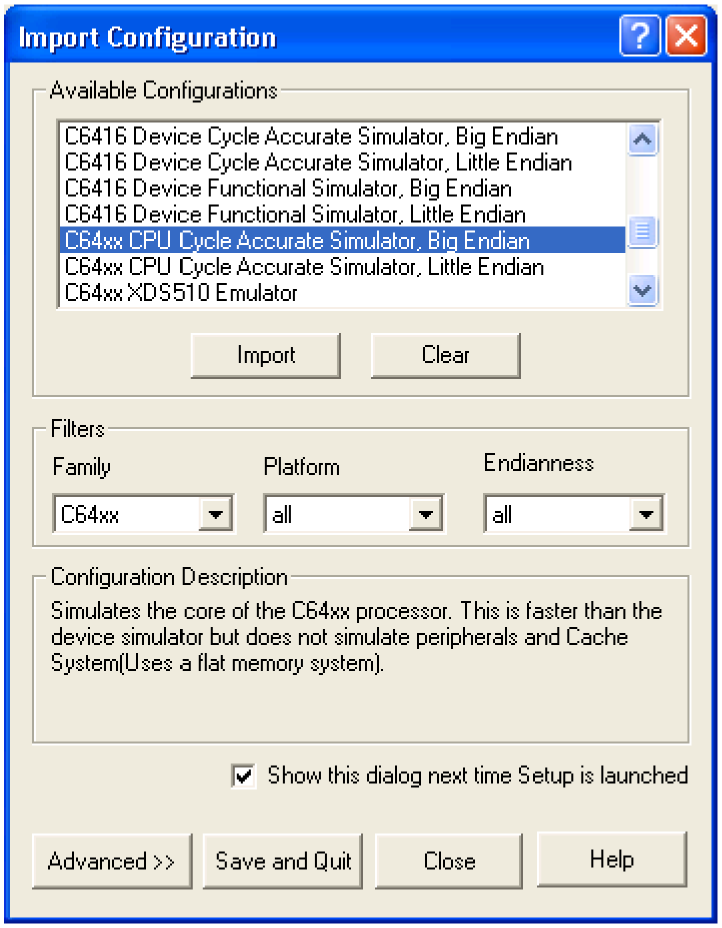
3 Application Design

3.1 Code Composer Studio Setup

Code Composer Studio Setup is a utility that is used to define the target board or simulator the user 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.

When running the IDE for the first time, a popup window informs you that you don't have a target configuration and prompt you to run the setup program. You may wish to change the system configuration to match your environment before 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).



C6416 Device Cycle Accurate Simulator, Big Endian
C6416 Device Cycle Accurate Simulator, Little Endian
C6416 Device Functional Simulator, Big Endian
C6416 Device Functional Simulator, Little Endian
C64xx CPU Cycle Accurate Simulator, Big Endian
C64xx CPU Cycle Accurate Simulator, Little Endian
C64xx XDS510 Emulator

Family: C64xx
Platform: all
Endianness: all

Simulates the core of the C64xx processor. This is faster than the device simulator but does not simulate peripherals and Cache System(Uses a flat memory system).

[✓] 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 the way you like, you can save your settings to a configuration file and share it with other developers. Saving files is also useful if you are using more than one configuration and must switch between them.

Installing Device Drivers. If you are using an emulator or development board from an eXpressDSP third party, you may need to install the device driver for this target in Code Composer Studio Setup. Some third 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 v.2 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 v.3. If you decide to use a custom board data file instead of one automatically generated by setup, information in the Processor Configuration tab won’t be used to create a board data file. But the IDE still uses this information to determine the number and types 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 developer can specify the number and type of processors on the processor configuration tab.

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 v.3 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

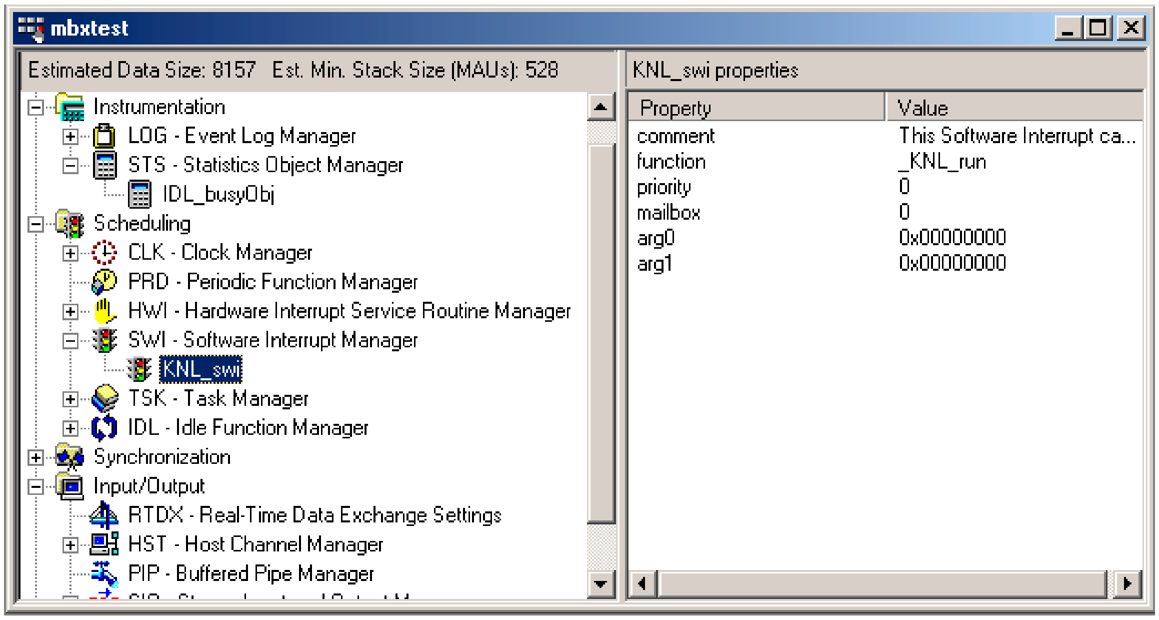
As DSP applications become more complex, it is important to structure them in an efficient, maintainable manner. This requires use of a real-time kernel that enables system functions to be allocated to different threads. The DSP/BIOS kernel provides an efficient set of kernel, real-time analysis, and 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. These capabilities make it easier to debug applications and optimize usage of OS and system resources.

DSP/BIOS consists of four parts: 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.



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 painstakingly calculate bitmaps.

3.3 TMS320 DSP Algorithm Standard

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 (called “TMS320 DSP Algorithm Standard” or “XDAIS”) reduces time-consuming system integration for anyone trying to include algorithms into their DSP system. As of June, 2004, more than 700 compliant algorithms are available on the TI’s third party network. Any of these algorithms can easily be integrated into applications 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.

XDAIS defines common programming rules and guidelines with a set of programming interfaces that are used consistently by algorithms across a wide variety of applications. The IDE includes key header files (i.e., ialg.h), several basic examples to illustrate use of the standard (e.g., FIR/FIG algorithms and clients) documentation about the the standard itself (SPRU352) and application notes. A demonstration project runs on most TI low cost DSP Starter kits (DSK). This project implements an audio player and recorder using compliant algorithms. To access this demonstration, visit **www.ti.com/dummiesbook**.

Several productivity tools assist in algorithm development and testing. A code generation wizard automates the generation of algorithm-compliant code, and a QualiTI test tool identifies whether an algorithm complies with the standard. To access these free tools, go to Help » Web Resources » Application Software or go to www.dspvillage.com and choose Software.

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.

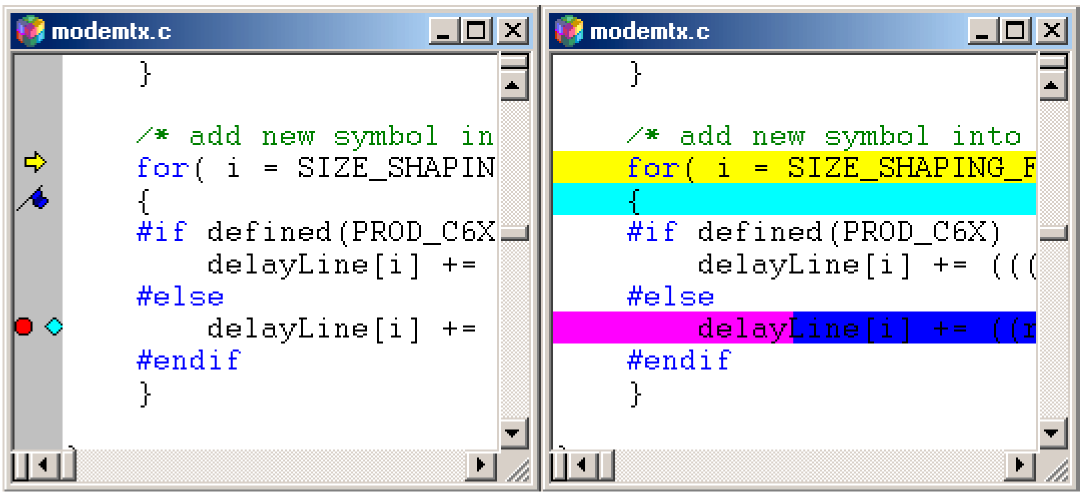
Bookmarks. Use bookmarks to find and maintain key locations within your source files. A bookmark can be set on any line of any source file. Bookmarks are saved with a Code Composer Studio workspace so that they can be recalled at any time.

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 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. In Figure 4, the right panel shows the editor with the Selection Margin turned on.



```
    }

    /* 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. The left panel shows the editor window with Selection Margin turned on

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 makes it possible to determine the actual instructions 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.

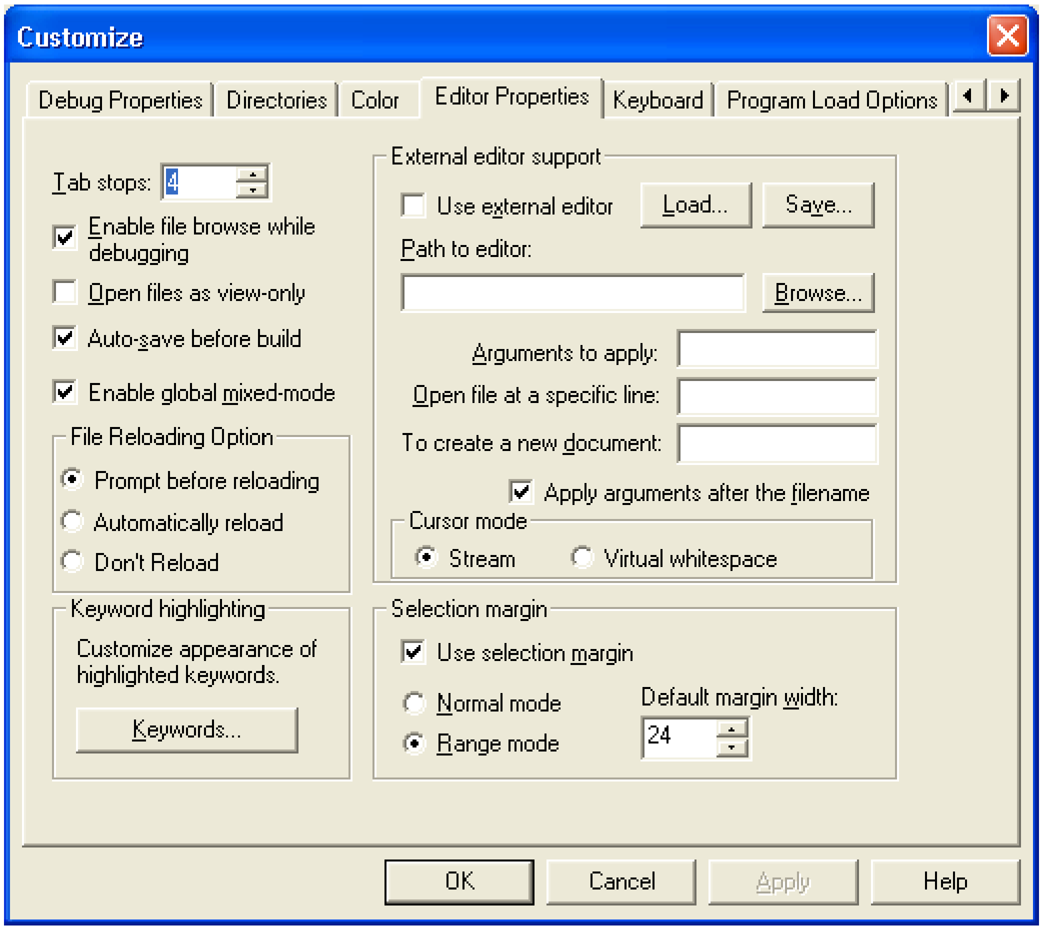
Custom Keywords. The integrated editor features keyword highlighting. Keywords, comments, strings, assembler directives, and GEL commands are highlighted in different colors. Keyword highlighting is available for C, C++, Assembly, and GEL files. When a source file is opened or saved, the editor recognizes 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:\CCStudio, the default keyword files can be found c:\CCStudio\cc\bin\Editor\Keywords\Base. Custom keyword files should be created in c:\CCStudio\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 prompts you to enable the default keyword highlighting feature.

4.1.1 External Editor Support

To configure an external editor for use with Code Composer Studio IDE, 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. The IDE includes default configuration settings for several popular editors which you can load and even customize. 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
[✓] Enable global mixed-mode

File Reloading Option
[✓ Prompt before reloading] Automatically reload / Don't Reload

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
Cursor mode
[✓ Stream] Virtual whitespace

Selection margin
[✓] Use selection margin
Normal mode [✓ Range mode]
Default margin width: 24

Figure 5. External Editor Settings

4.2 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 with 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. The project manager automatically scans dependencies, making it unnecessary to add dependent files manually to a 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 build options for each project configuration.

4.2.1 Project Configurations

The Code Composer Studio Project Manager supports multiple build configurations. Multiple build configurations allow (for example) different sets of build options without requiring the user to make changes to project options. Folder names match configuration names to make it easy to track where configuration−specific output files go. Build configurations can be saved and later retrieved for bit exact rebuilds.

4.2.2 Multiple Project Support

It is not unusual for a developer to work on several projects at once. For instance, a developer may have an old project to maintain, a project that represents the next product release and a test project. The Code Composer Studio Project Manager allows you to keep multiple projects open simultaneously and switch between them easily. Larger more complex applications, for example, might consist of several library subprojects, each with separate project files.

4.2.3 Source Control

Code Composer Studio IDE lets developers use 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. That includes Rational’s ClearCase, Microsoft’s SourceSafe, Starbase StarTeam PVCS or MKS. Once the IDE connects to the source code control application, the project visually displays checkout status. This integration simplifies the tasks of project management, revision control, branching, merging and backups.

The project view window lists all source files in a project’s source folder. Files that are ‘checked out’ to the source control system 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 are indicated by red checks to the left of the name. The user only has read−only access to “checked in” files.

Once the IDE is configured for a specific source control provider, the IDE will invoke the provider’s tools for source control tasks. The IDE’s source control features might vary according to the capabilities of the provider.

4.2.4 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. Although both types of projects offer a multitude of build options for the developer, only executable projects allow the setting of linker options. Instead of this, library projects allow the setting of archiver options.

4.2.5 Custom Build Steps

Custom build steps help to automate 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. For example, you may wish 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:\CCStudio.

4.2.6 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 is called from within Code Composer Studio IDE, giving the effect of a fully integrated build environment. To launch the make process, the user will need to specify the command to build the makefile and define how to parse source files. This parsing is done with 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.

4.2.7 Exporting Makefiles

In some cases, the developer will need to create an application for other environments or operating systems (such as Unix). A developer can create the application using Code Composer Studio IDE and export the project to a standard makefile (by selecting Project ›Export to a Makefile). This allows the developer to do final builds in other operating systems.

4.2.8 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 calls 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.

4.2.9 Project Dependencies

Creating subprojects or dependency relationships between projects is often helpful for debugging. For example, you can use dependent projects to group projects with the same configuration settings. Or you can try one configuration with a particular subproject and another configuration without it for comparison purposes. To add a project dependency, drag an open project over another open project in the project window. That will cause a new directory icon, Dependent Projects, to appear underneath the root project. You can switch project configurations for a subproject by double-clicking underneath the Setting column within that same dialog window.

4.3 Code Generation Tools

In the past, developing high performance DSP code has required the developer to optimize assembly code by hand and 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.

4.3.1 C6000 Code Generation Tools

For the embedded software developer, the C6000 Compile Tools — co-developed with the DSP architecture — offer top performance with global view analysis and architecture-specific optimizations such as interactive profiling, tuning and feedback.

One major 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. C6000−specific enhancements include software pipelining, data path partitioning, inner and nested loop optimization, unrolling, and predication.

5 Debug

5.1 Debugger

The Code Composer Studio debugger makes it easy to find and fix errors in real−time embedded applications. For example, 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 Window. 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 functions 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*

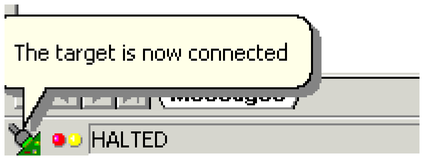
A Probe Point™ can be set at any point in your algorithm (similar to the way a breakpoint is set). When the program reaches a Probe Point, the executing program updates the connected object (whether it be a file, graph, or memory window). After updating, execution will resume. Probe Points can be enabled or disabled, 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 known set of resultant values.

5.2 Connect/Disconnect

Now it is easier to connect to and disconnect from a target dynamically (See Figure 6). Not only can you connect or disconnect on the fly, you can even restore previous debug states when reconnecting (by choosing the "Restore Debug State" under the Debug menu). Selecting this option enables every breakpoint that was disabled at last disconnect. The status bar briefly flashes a message to indicate changes in a target's status. It also indicates the last known execution state of the target and (if connected) whether the target is stepping and what kind of breakpoint caused the halt.


The target is now connected

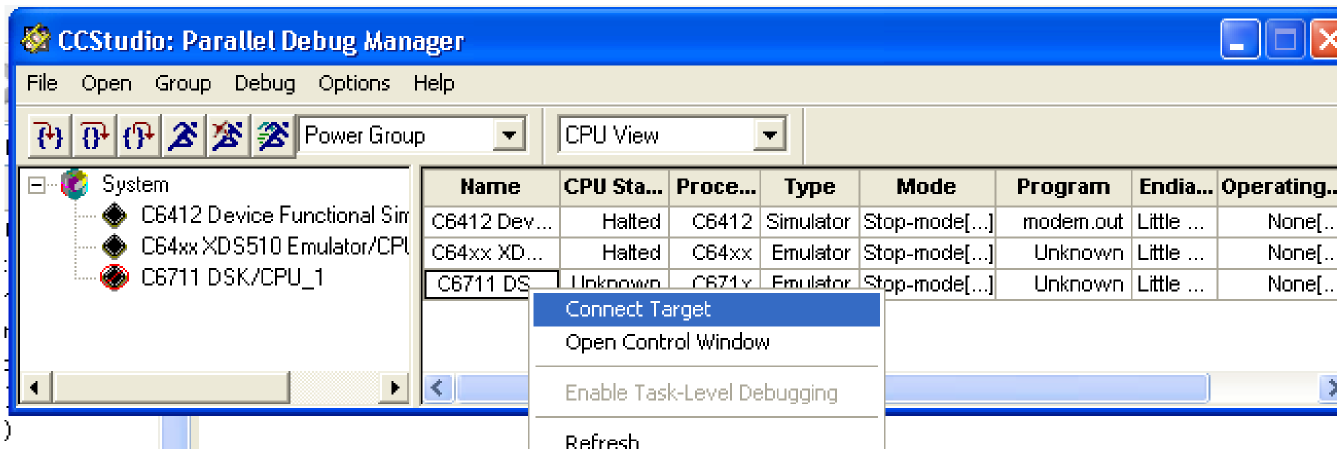
HALTED

Figure 6. The Status Bar Indicates When the Target is Connected or Disconnected

5.2.1 *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. You can broadcast commands to a group of processors.

The visual interface of PDM allows the user to create custom groups of targets and view processors by group membership. The user can put processors into tightly-coupled groups (where processors are on the same physical scan chain) and loosely-coupled groups (i.e., where the processors are not all on the same scan chain).


CCStudio: Parallel Debug Manager
File Open Group Debug Options Help
Power Group
CPU View
Connect Target
Open Control Window
Enable Task-Level Debugging
Refresh

System
- C6412 Device Functional Sim
- C64xx XDS510 Emulator/CPU
- C6711 DSK/CPU_1

Name	CPU Sta...	Proce...	Type	Mode	Program	Endia...	Operating..
C6412 Dev...	Halted	C6412	Simulator	Stop-mode[...]	modem.out	Little ...	None[..
C64xx XD...	Halted	C64xx	Emulator	Stop-mode[...]	Unknown	Little ...	None[..
C6711 DS	Unknown	C671x	Emulator	Stop-mode[...]	Unknown	Little ...	None[..

Figure 7. PDM Lets You Create Custom Groups of Targets and View Target Status

Synchronized Commands. It is possible to broadcast commands via PDM to all target processors in a tightly-controlled 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.

Viewing Connection Status from PDM. PDM provides an easy way to view the status of each target while the user is working in the IDE. For example, one can see the mode, CPU status and endianness of a target; one can also use the right-click menu to connect to the target or the dropdown menu to change an option for a target. One can also connect or disconnect dynamically to this target using PDM.

5.3 Real-Time Data Exchange (RTDX)

At one time DSP developers were able to exchange data snapshots with the host computer by using breakpoints to stop their application. The use of these data snapshots, (called ”stop–mode debugging”) often failed to provide an accurate view of a system’s real world operation.

RTDX offers one solution to this problem. This technology (invented by Texas Instruments) gives designers the industry’s first DSP system to provide real–time, continuous visibility about how target applications actually perform. RTDX allows developers to transfer data between the host computer and DSP devices without stopping their target application. RTDX allows designers to monitor continuously their systems and gain real–time insights into running applications. It also allows data to be streamed with ActiveX–compliant application such as Excel, LabVIEW or MATLAB. When displaying RTDX data, host applications can read either live or saved data. The IDE also supports RTDX with multiple processors on either the same or different scan paths. RTDX has tools for diagnosis and configuration as well as a channel viewer for detecting and viewing target–declared channels. Developers can also run RTDX with the instruction simulator, allowing RTDX to be used before target hardware is available.

You can use High-speed RTDX (HSRTDX) to provide faster data logging rates and increased responsivness for RTDX-enabled host controlled applications than traditional RTDX. HSRTDX requires a HSRTDX-enabled DSP and an XDS560 class emulation controller.

5.4 Advanced Event Triggering (AET)

Advanced Event Triggering is designed to simplify hardware analysis. 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 conditions and actions. When you run the target program, the sequence program looks for the condition and performs the action you requested.

6 Application Code Tuning

6.1 What It Does

The Code Composer Studio IDE now offers a series of tools (called Tuning Tools) to help developers to optimize applications. These tools allow users to set goals and track progress towards goals by collecting different sets of data. A new Advice Window provides on-the-fly recommendations about how to make code meet user goals.

The IDE has two Layouts, (tuning layout and standard layout), which can be toggled from the main toolbar. Standard layout is the default, while tuning layout (represented by a tuning fork icon on the toolbar) opens advice windows on the left side. The tuning layout focuses attention on the optimization needs of the user’s program. The advice window walks the user through the optimization/tuning process specific to the DSP device and a selected goal.

Changing the layout does not require having to close or save your work. When you switch layouts, the Control Window momentarily disappears and reappears in the new layout. Layouts mainly affect the way windows are displayed in the Code Composer Studio IDE.

6.1.1 Tuning Dashboard

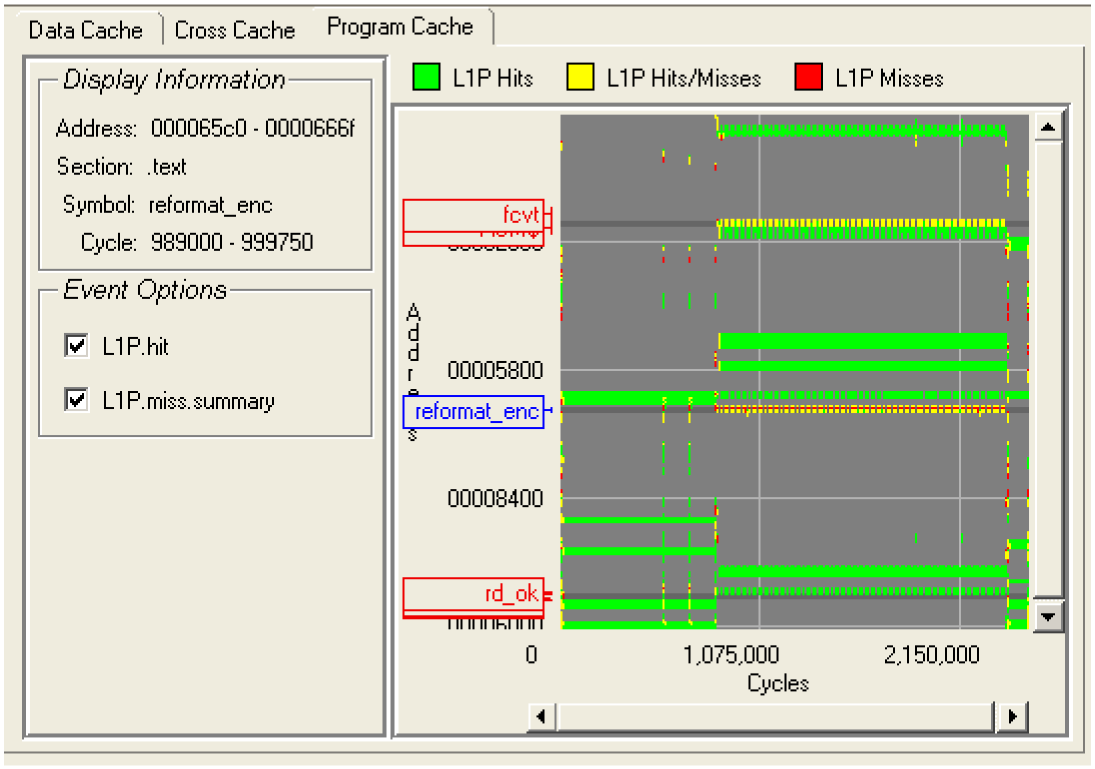
The Tuning Dashboard panel comes up on the left when the user switches to Tuning Layout (by selecting the Tuning Fork icon on the toolbar). This panel provides advice and allows the user to launch tuning tools (i.e., Advice Window, Goals Window, Profile Setup, and Profile Viewer). Tuning Tools can also be launched from the Profile menu.

6.1.2 Compiler Consultant

After analyzing source code and compiler build options, the Compiler Consultant recommends ways to improve code performance. The tool displays two types of information: Compile Time Loop Information (created by the Compiler) and Run Time Loop Information (gathered by profiling the application). Each time the application is compiled, Compiler Consultant analyzes the code and creates suggestions for different optimization techniques to improve code efficiency These suggestions include compiler optimization switches and pragmas to insert into the code that give the compiler more application information.

6.1.3 CacheTune

By graphically representing cache memory accesses, CacheTune makes it easier to identify non-optimal cache usage. This display of cache hit/miss data as a two-dimensional graph can be used to improve performance in cache-based memory devices. All memory accesses are color-coded by type. Various filters, panning, and zoom features facilitate quick drill-down to view specific areas. This visual/temporal view of cache accesses enables quick identification of problem areas, such as areas related to conflict, capacity, or compulsory misses. All of these features help the user to improve an application's overall cache efficiency.



Address: 000065c0 - 0000666f
Section: .text
Symbol: reformat_enc
Cycle: 989000 - 999750

✓ L1P.hit
✓ L1P.miss.summary

Figure 8. CacheTune Uses Colors to Distinguish Cache Hits From Misses. It Graphs Symbols That Conflict in Memory.

6.1.4 CodeSizeTune

CodeSizeTune (CST) replaces the Profile-based Compilation (PBC) tool from version 2.2 of the IDE. CST analyzes tradeoffs between code size and cycle count more quickly and easily than before. Using a variety of profiling configurations, CodeSizeTune profiles the user's application, collects data on individual functions, and determines the best combinations of compiler options. CST graphs function-specific options sets, permitting a choice of the configuration best suited for the user's needs.

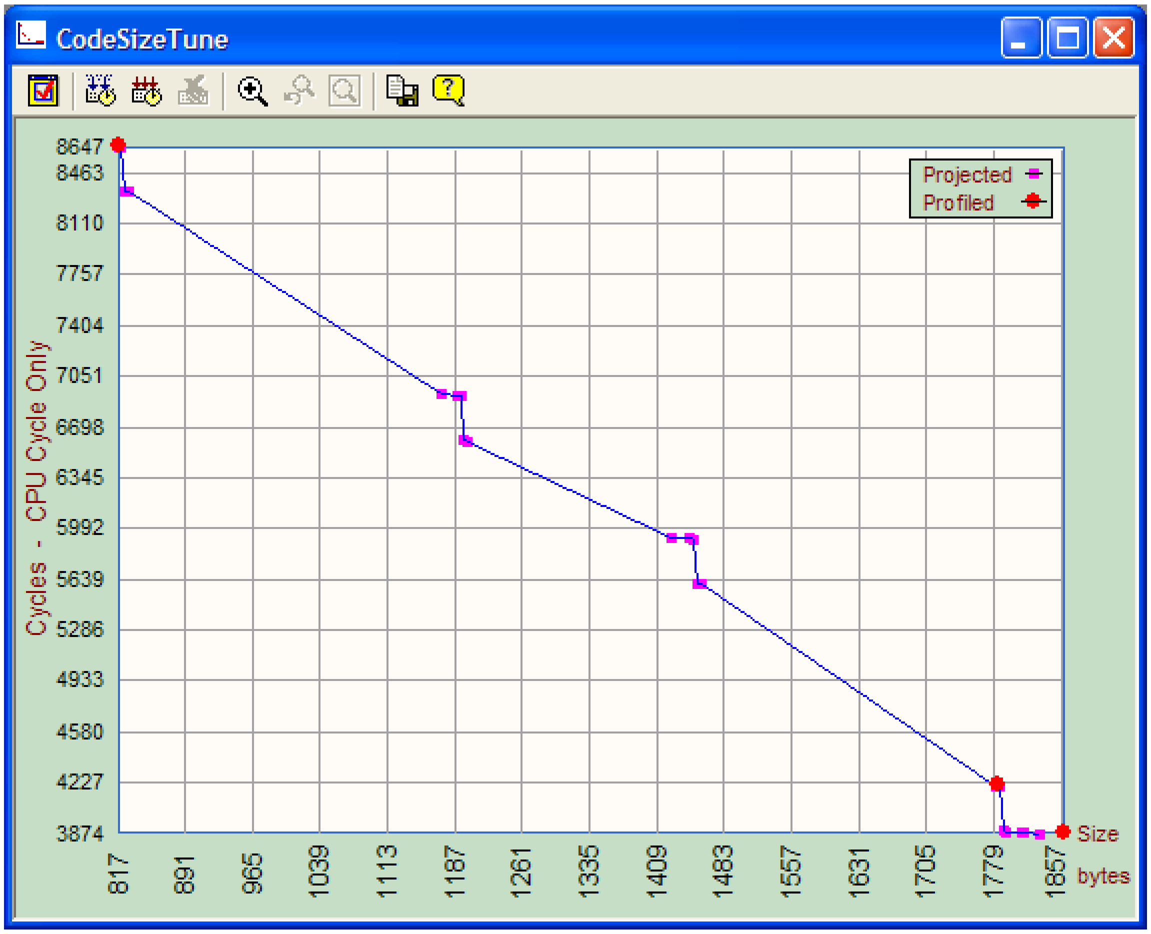


Figure 9. CodeSizeTune lets you plot code size vs. performance. It is part of the Tuning Tools for Optimizing Code

7 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 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.

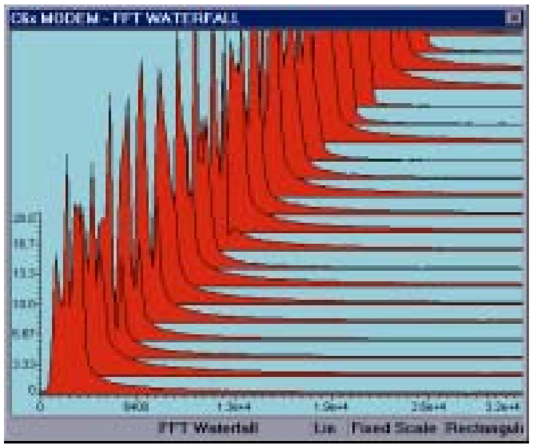


Figure 10. FFT Waterfall

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).

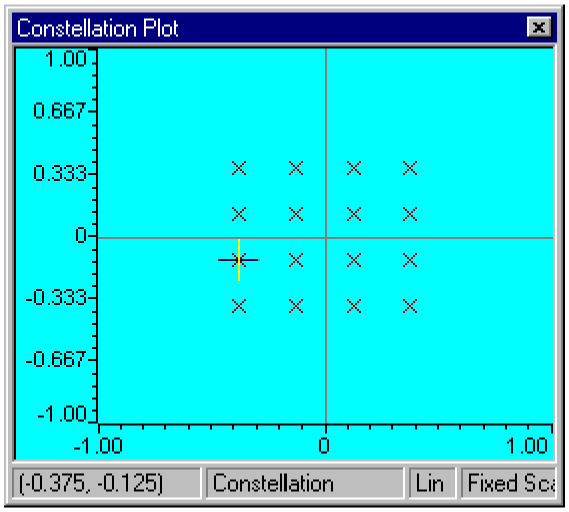


Figure 11. Constellation Plot

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.

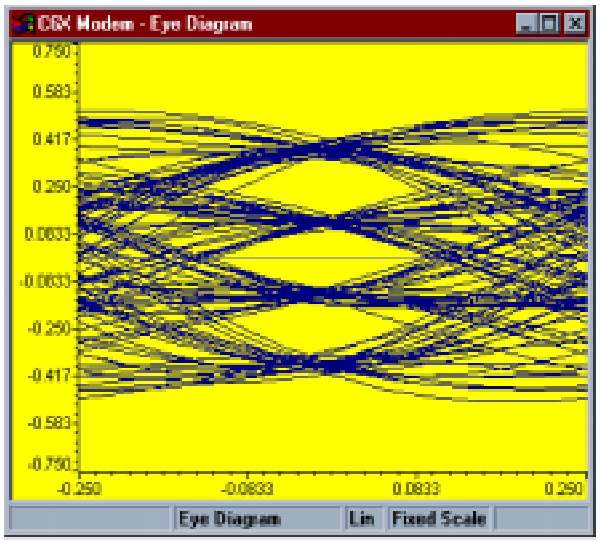


Figure 12. Eye Diagram

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

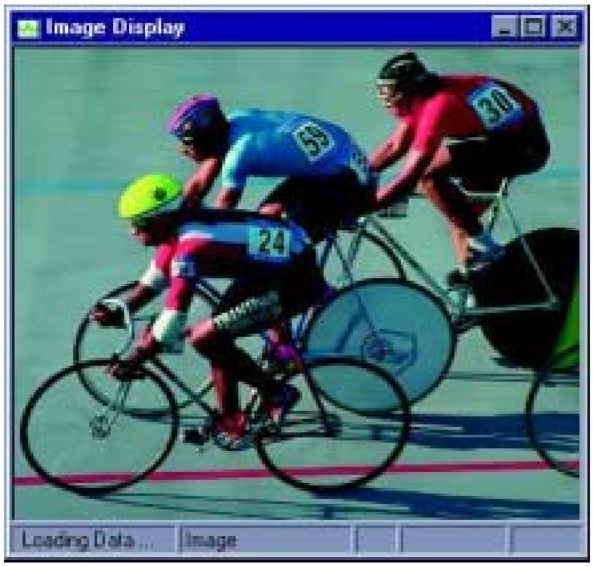


Figure 13. Image Display

8 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 help developers to see time-related interactions between code sections. They can reveal subtle real-time problems that otherwise might go undetected. 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, it can be fixed'.

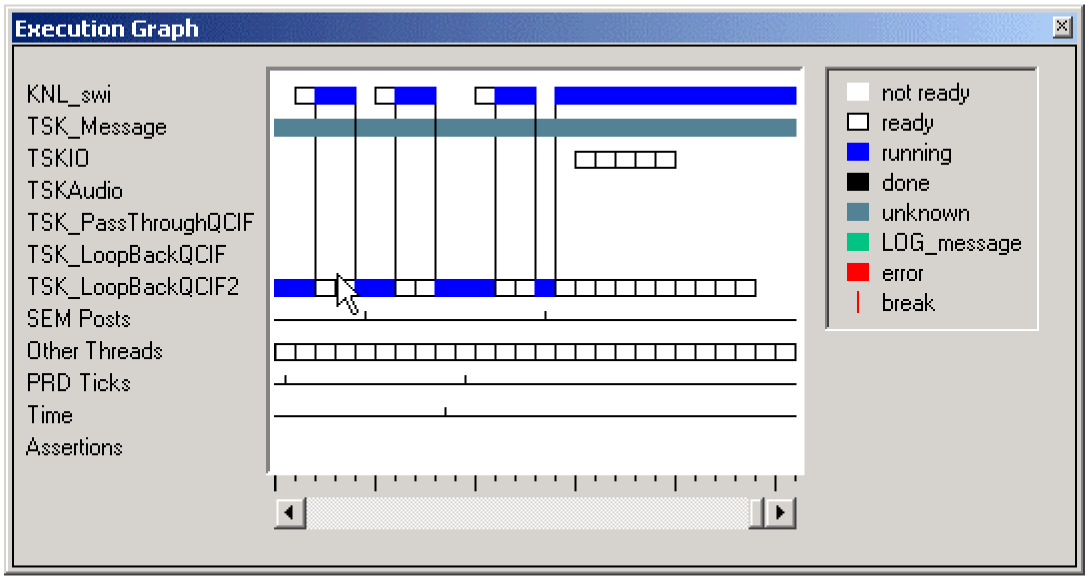


Figure 14. Find and Fix Subtle Real-Time Problems

Debug and analysis can be 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 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 can serve as a foundation for a new generation of manufacturing and field diagnostic tools.

9 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. After creating GEL functions using the GEL grammar, you can load them into Code Composer Studio IDE. With GEL, you can access actual/simulated target memory locations and add options to the 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 that 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.

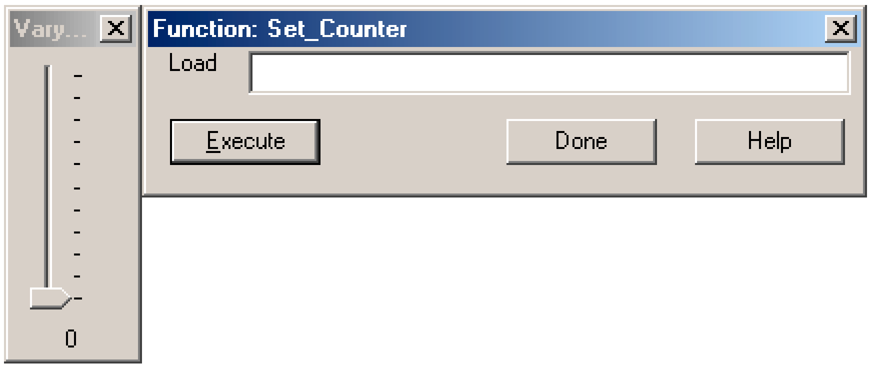


Figure 15. GEL Controls

10 Summary

Code Composer Studio IDE v3.0 represents the evolution of the DSP development environment. It contains functionality needed by today's larger, distributed, global project teams. By reducing time spent on repetitive tasks and tool development, the IDE gives the developer more time to focus on innovation.