Application Report
SPRAA42 – September 2004

Using CodeSizeTune to Optimize Your Application's Performance and Code Size

Ruby Li *Software Development System*

ABSTRACT

Today, DSP developers often find it difficult and time consuming to find the compiler options that optimize both speed performance and code size. In most cases, when performance increases, so does code size. The performance constraints and code size constraints are often at odds. Previously, to find the compiler options with an optimal balance of performance and code size, developers had to build and profile their application repeatedly, each time using a different set of compiler options. With a large application, it was not possible to try all compiler option combinations at the function level. CodeSizeTune (previously known as Profile Based Compilation (PBC)) now automates this process for developers of the TMS320C6000™ family of DSPs. CodeSizeTune displays the possible code size and performance (cycle count) tradeoffs in a graphical form. Developers can easily pick one or more desired solutions from the graph and save them as different configurations in the project, thus providing an optimal solution and speeding up the time-to-market.

Contents

Figures

1 Introduction

CodeSizeTune (CST) is a tool to help DSP developers choose the best compiler options to optimize their application's code size and performance. To achieve this goal, CST automates the process of building and profiling the application using different compiler options that have different levels of code size and performance optimization. CST then considers all combinations of compiler options for each function and calculates their code size and cycle count results.

In this application note, we compare the code size and cycle count before and after running CST. The example used shows how to reduce code size by 25% without losing any performance.

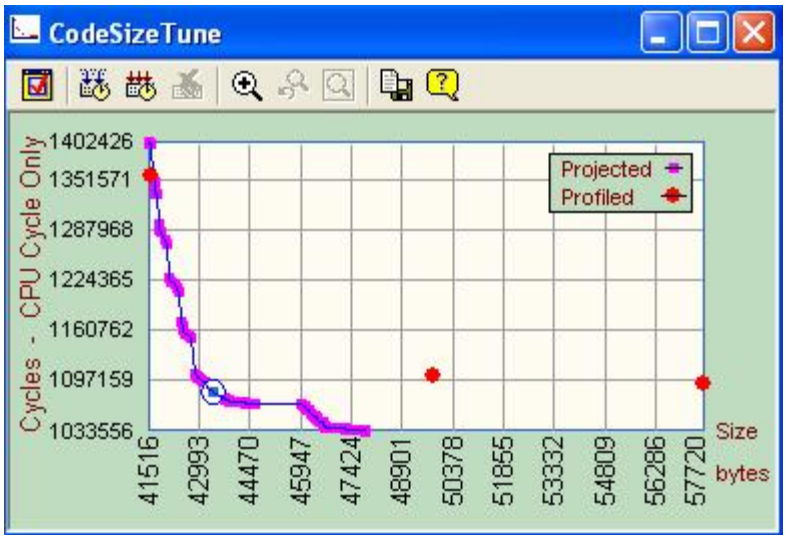

Figure 1. Example of a CST Graph

To let developers easily find the solution for their application, code size and cycle count (performance) results for each build are profiled, and each possible solution is plotted in a graph as a point as shown in Figure 1. A set of points represents all possible tradeoffs between code size and performance of the given application. Developers can then easily pick a point from the graph that best meets their code size and performance constraints. The user can also view the function-level code size results, cycle count results, and the compiler options in a table as shown in Figure 2.

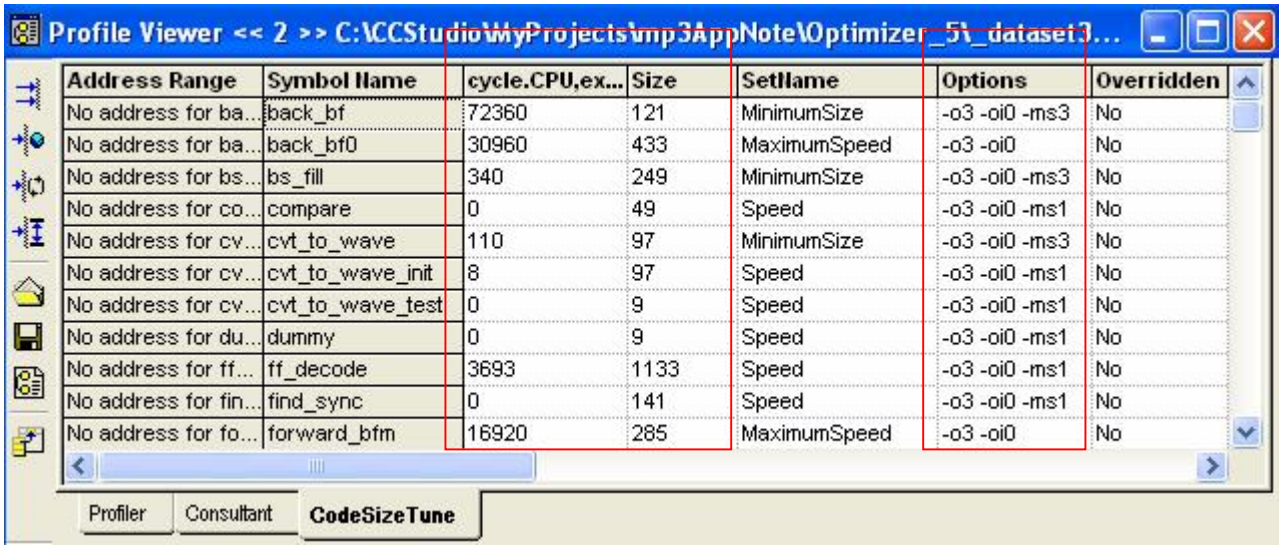

Figure 2. Example of a Table Showing Profile Results

In some cases, the developer may want to directly specify which options set is assigned to a particular function rather than let CST select based on its calculations. Perhaps the function was not invoked during the profile session (for example, it may be an interrupt handler), or perhaps the input data read during the profile session was not representative. The developer can override the compiler options that CST assigns to a function using the Overrides window. The override functionality is discussed in Section 5.1.

Finally, the developer can save any desired solution points from the graph to different build configurations in the project. By rebuilding the application with the saved configuration, users know that the code size and performance have been optimized for their needs. If desired, developers can further modify the function-level compiler options after they finish using CST.

2 Development Flow

Use the following development flow to get the best results from CodeSizeTune within the new tuning environment (see Figure 3). Finish the debug phase before starting the tuning phase.

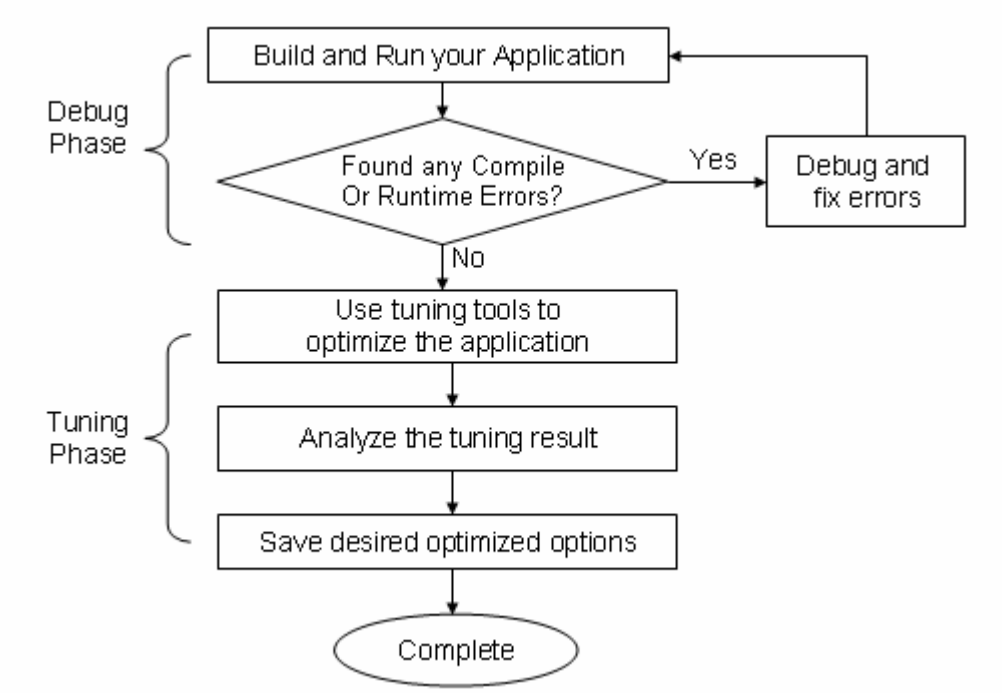

Figure 3. Code Development Flow

This document focuses on using CST in the tuning phase. For more information about the general tuning environment and other tuning tools, refer to topic Application Code Tuning in the Code Composer Studio™ online help document.

3 Supported Platforms and Expected Use

CodeSizeTune is available in the Code Composer Studio™ Integrated Development Environment (IDE) v3.0 for all C6000™ simulators. CST supports simulators only. It does not support library projects, but it does support library subprojects. If the project contains any executable subprojects, CST does not make tradeoffs for the subprojects, and ignores them. CST is a Code Composer Studio™ IDE plug-in. The developer is expected to use CST through the Code Composer Studio™ IDE. The user must have a Code Composer Studio™ IDE project to use CST. If no project is loaded, the CST menu item is disabled as shown in Figure 4.

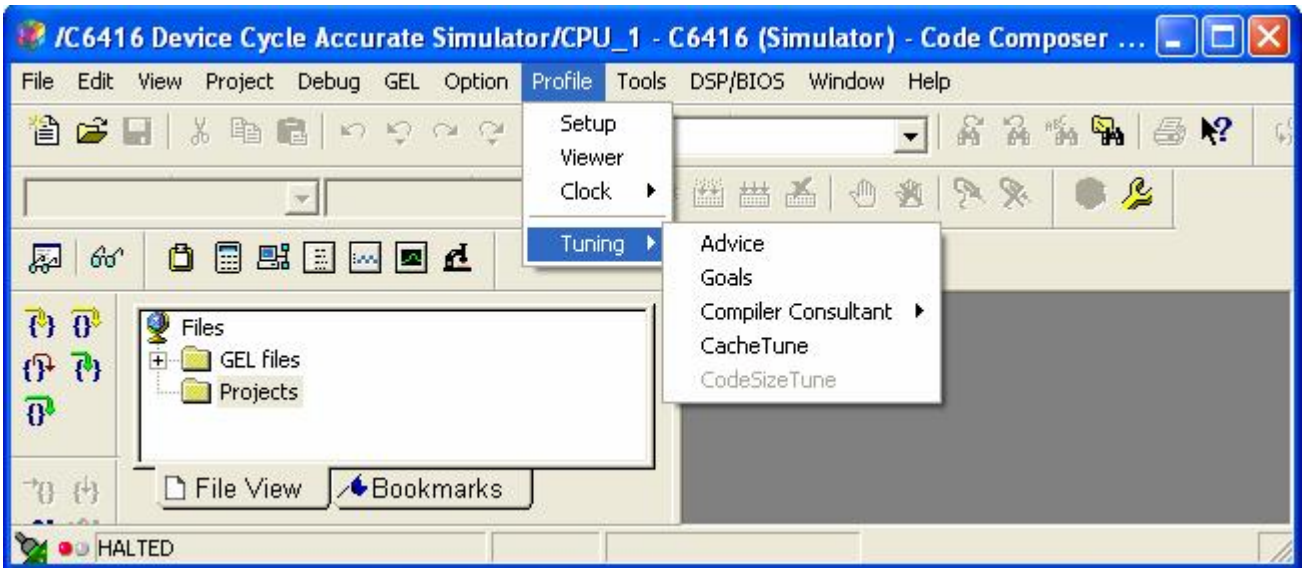

Figure 4. CST Menu Item Is Disabled When no Project Is Loaded

CST assumes that the application is set up for profiling, and therefore expects the application to run after functionality has been tested and no compile or runtime errors exist. This is essential since the application runs in a batch program (non-interactive with the user) to extract profile/tuning information. The application must self-terminate (does not run in an endless loop) or have exit points established through the profile setup. An exit point tells the profiler to stop collecting profile data. For information about creating and using exit points, refer to topic Application Code Tuning -> Dashboard – Profile, Goals, Advice -> Profile Setup -> How to…->To Create an Exit Point, in the Code Composer Studio™ online help document. CST assumes that the application should use input that is representative of typical usage and should be the same across all profiles. Input can be data, interrupts, and other stimuli that affect the application.

4 An Example Using CodeSizeTune

4.1 About the Example

This example demonstrates how to use CodeSizeTune to reduce code size by 25% on the TMS320C6416 without sacrificing the performance of the application. It shows how to select and save the best compiler options to optimize both the code size and cycle count (performance) of an application. The project MP3.PJT illustrates how to reduce code size by 25% without losing any performance. The screen captures shown below are from the C6416 device simulator, little endian. To keep the length and content of the example manageable, this example focuses on CST usage. Therefore, this application note assumes that you are familiar with the fundamental operations of Code Composer Studio™ IDE, that is, installation, setup, creating a project, and loading a program.

NOTE: The MP3 code provided here is for educational purposes only; therefore, neither runtime nor production license is granted (nor should be inferred).

4.2 Code Composer Studio™ IDE and Profile Setup

4.2.1 Code Composer Studio™ IDE Setup

The first step is to select a simulator. After Code Composer Studio™ IDE is installed properly, select the desired simulator configuration through the Code Composer Studio™ IDE setup utility. This example uses the C6416 device simulator (little endian), which provides cycle-accurate program execution time.

Launch Code Composer Studio™ IDE and open the application with the Code Composer Studio™ IDE main menu selection Project -> Open. Use the MP3 project that accompanies this application note. CodeSizeTune works on a per-project basis. Note that if there is more than one configuration in the project (for example, Debug and Release), CST works with the active configuration only. Ensure that the active configuration is set to the one you want CST to tune. This example uses the Debug configuration. Check the active build configuration with the main menu selection Project -> Configurations….

Once you have the project open, ensure that the project has passed the debug phase before you go into the tuning phase. The project must have no compile or runtime errors.

4.2.2 Profile Setup (Optional)

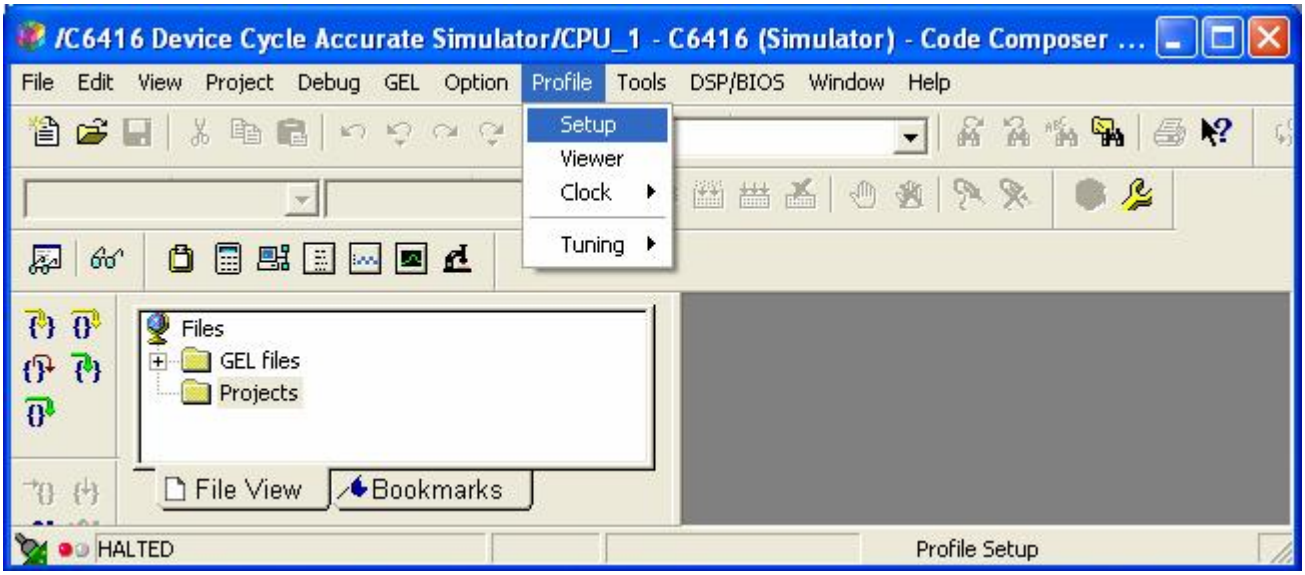

Figure 5. The Menu Item to Open the Profile Setup Utility

You do not have to use the profile setup utility (see Figure 5) if you accept all of the CST defaults. By default, CST sets no exit point in your application. CST runs the application from start to end. The user can set one or more exit points in an application through the Control tab of the profile setup utility. For information about how to create an exit point, refer to topic Application Code Tuning -> Dashboard – Profile, Goals, Advice -> Profile Setup -> How to…->To Create an Exit Point, in the Code Composer Studio™ online help document.

Note that CST collects information using the *.cycle CPU: Excl. Total* event. CST information does not include any system effects such as cache misses, off-chip memory access, memory bank conflicts, etc. The user cannot change the event that CST uses to collect information.

The MP3 project in this example self-terminates. The example compares the total code size and cycle count of all functions. Thus, the example uses the CST defaults and sets no exit point or profile range in the profile setup.

4.3 Getting the Total Code Size and CPU Cycle Count Results Before Running CST

This section demonstrates how to get the code size and cycle count results before running CodeSizeTune. Because CST does not collect the runtime support (RTS) library profile data, and Code Composer Studio™ IDE does not show the total code size excluding the RTS library code size, we have modified the linker command file (lnk.cmd) in this example to separate the code size of the RTS library from the total code size. We have added the line *.rtstext > PMEM* in the SECTIONS area (see Figure 6).

```
SECTIONS
{
    .text       >       PMEM
    .rtstext    >       PMEM
    {
        -lrts6400.lib(.text)
    }
    .stack      >       PMEM
    .bss        >       PMEM
    .cinit      >       PMEM
    .cio        >       PMEM
    .const      >       PMEM
    .data       >       PMEM
    .switch     >       PMEM
    .sysmem     >       PMEM
    .far        >       PMEM
}
```

Figure 6. Modified Linker Command File

NOTE: By default, all program code is placed into a section called *.text*. By modifying the linker command file as described above, the code from the RTS library is put into a separate section called *.rtstext*.

With this technique, the RTS library code size is excluded, and we can compare the code size of the same number of functions with CST. The code size result is shown in the map file. Thus, make sure that the project has specified the map file name under the Linker tab of the Build Options dialog (see Figure 7). Launch the Build Options dialog with the Code Composer Studio™ IDE main menu selection Project->Build Options. After checking the linker command file and the map filename, the project is ready to be built and thus get the total code size, excluding the RTS library code size.

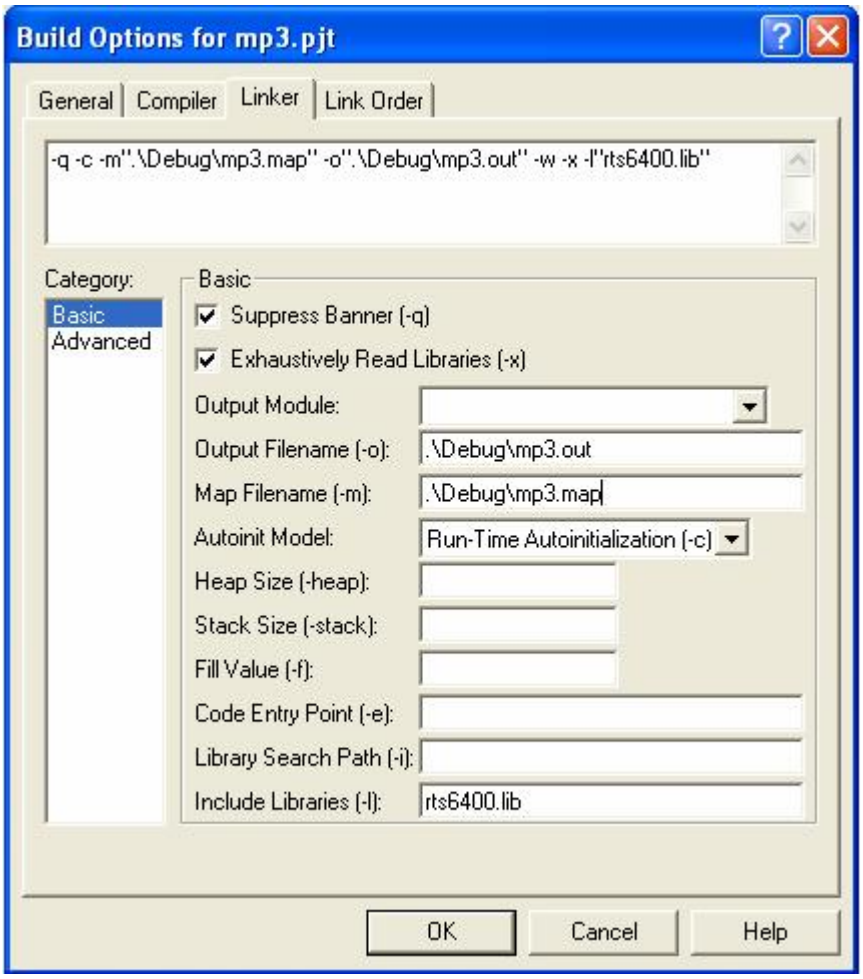

Figure 7. Map Filename in the Build Options Dialog

The next step is to build the project and load the program. Build the project with the Code Composer Studio™ IDE main menu selection Project -> Rebuild All. When the build process is done, load the program with the main menu selection File -> Load Program…. This example loads the program from the location *\MyProjects\mp3\Debug\mp3.out*.

NOTE: After the program is loaded, remember to set up the profiler.

Because CST collects CPU cycles only, set up the profiler to collect the CPU cycles for all functions. Remember to set up the profiler after the program is loaded. Now, launch the profile setup with the Code Composer Studio™ IDE main menu selection Profile -> Setup. Select the profile-setup toolbar button Enable/Disable Profiling and the button Enable/Disable All Functions to enable the profiler that profiles all functions (see Figure 8). Next, click the Custom tab, and check cycle -> CPU under the Range area as shown.

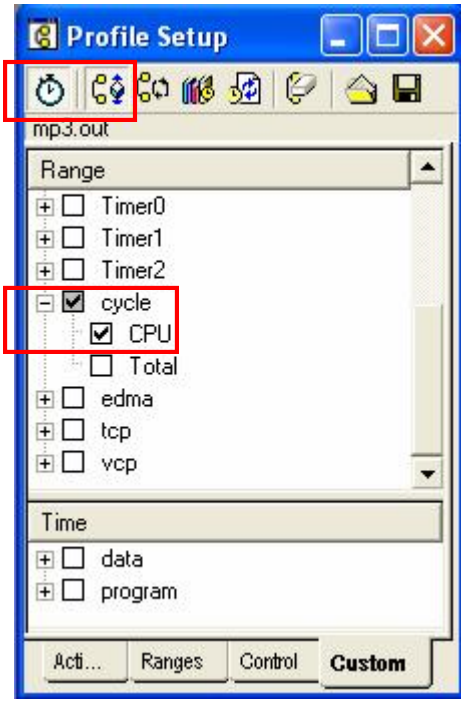

Figure 8. Profile Setup Window

The project is now ready to collect the CPU cycle count for all functions. Run the program with the main menu selection Debug -> Run. When run is complete, open the profile viewer to see the cycle count result under the Profiler tab as shown in Figure 9. The profile viewer does not show the summation of the cycle counts. Therefore, add up all the numbers under the *cycle.CPU:Excl.Total* column manually or export the table to a spreadsheet. The total CPU cycle count for this project is 1,090,758. Your cycle count result may vary.

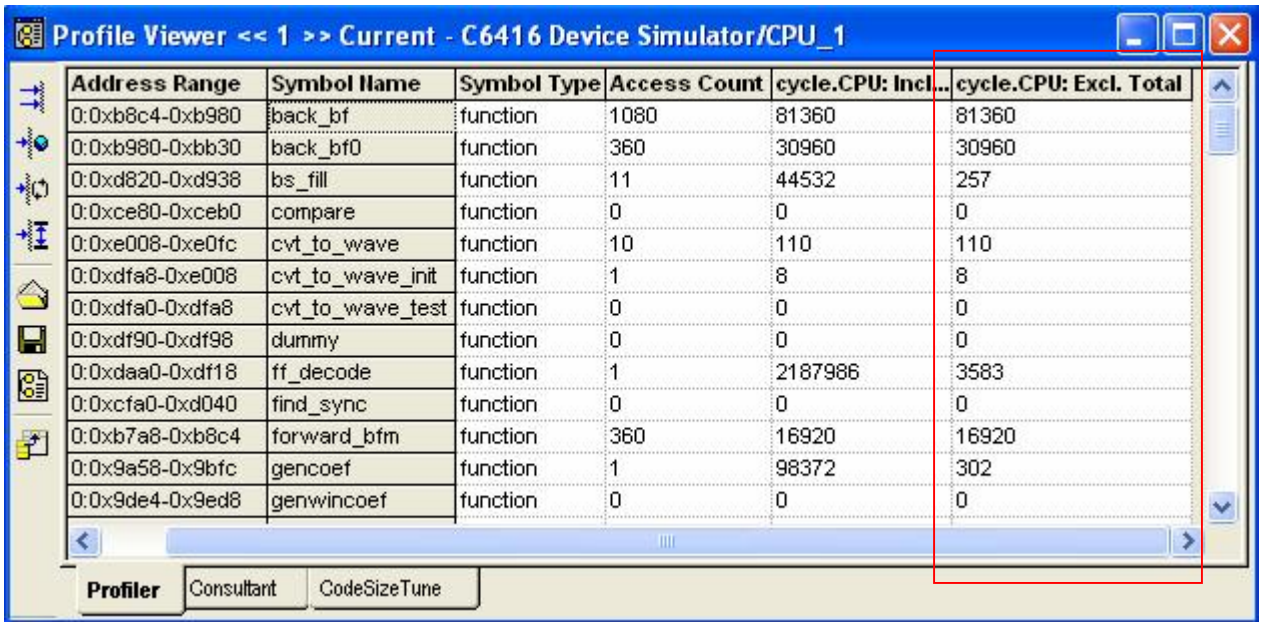

Figure 9. Profile Viewer Window

See the total code size by opening the map file either within Code Composer Studio™ IDE or through any text editor. Scroll down to the *.text* section in the map file. The total code size is shown under the length column of the first row (see Figure 10). This code size result does not include the RTS library code size, and we can compare the code size of the same number of functions with CST. The total code size for this project is 0x0000e0e0 or 57,568 in decimal.

```
SECTION ALLOCATION MAP

 output                                  attributes/
section   page    origin      length       input sections
--------  ----  ----------  ----------   ----------------
.data       0    00000020    00000000     UNINITIALIZED

.text       0    00000020    0000e0e0
                   00000020    00005e20     iup.obj (.text)
                   00005e40    00003b60     iwinm.obj (.text)
                   000099a0    00001d40     isbt.obj (.text)
                   0000b6e0    000017a0     icdct.obj (.text)
                   0000ce80    000009a0     mhead.obj (.text)
                   0000d820    00000780     towave.obj (.text)
                   0000dfa0    00000160     wavep.obj (.text)

.rtstext    0    0000e100    0000c380
```

Figure 10. MP3 Map File Showing the Total Code Size

The next section demonstrates how to run CST to get the optimized code size and cycle count results.

4.4 Tuning Code Size Using CST

4.4.1 Launching CST

Launch CST with the menu selection Profile -> Tuning -> CodeSizeTune (see Figure 11).

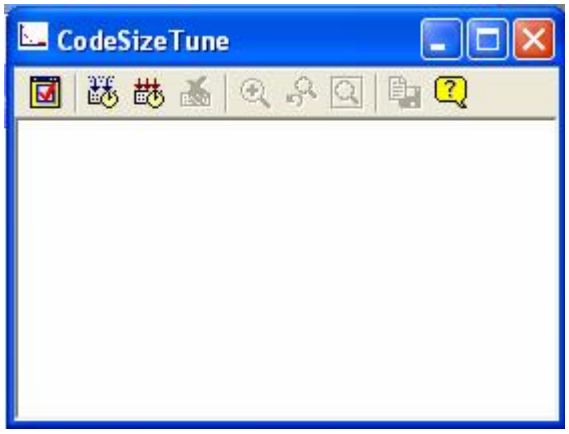

Figure 11. Initial CST Window

NOTE: If you hold your mouse pointer over a CST toolbar button, the respective tool tip displays the short description of that toolbar button.

When the CST window is launched, the advice window with the first step of CST advice is also launched (see Figure 12).

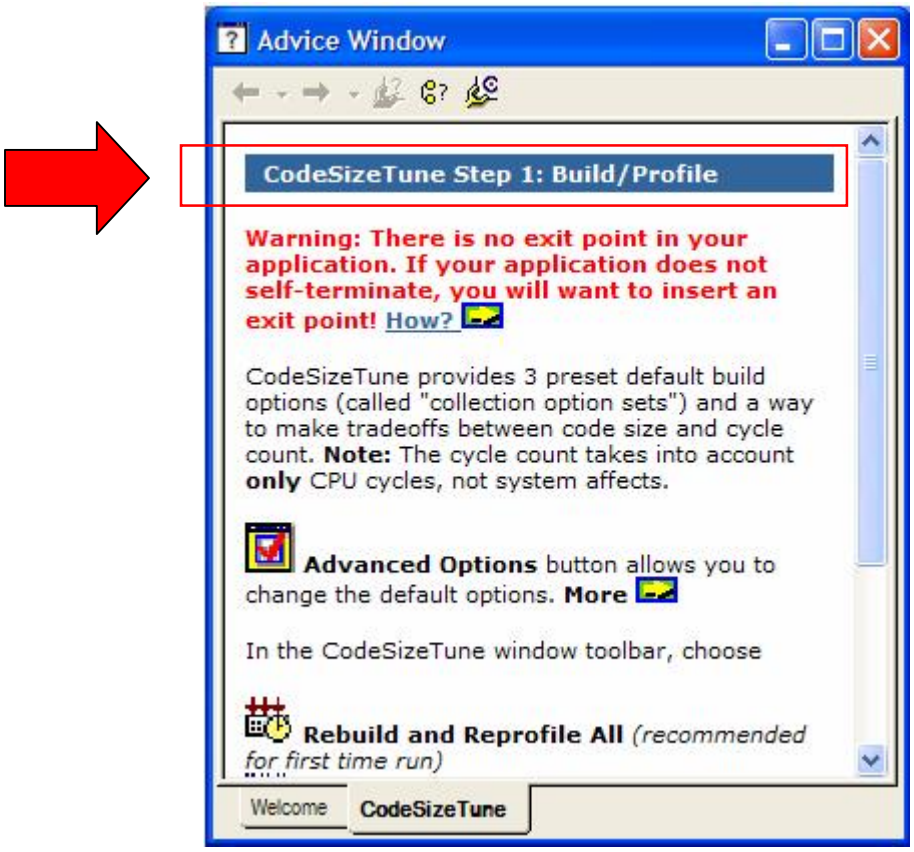

Figure 12. The First Step of CST Advice in the Advice Window

4.4.2 Building and Profiling

The first step of the advice is to build and profile the application using CST. CST, by default, selects three collection options sets with varying degrees of code size vs. speed to build and profile the application. A collection options set is a CST preset default, build-options set. View and change the defaults with the toolbar button selection Advanced Options -> Collection Options…. This launches the Choose Profile Collection Options dialog as shown in Figure 13. The dialog shows which compiler options are used for each CST collection options set. The more collection options sets you select, the longer it takes CST to build and profile. You can create new collection options sets through this dialog. For more information on how to create a new collection options set, refer to topic Application Code Tuning -> CodeSizeTune -> Advanced Usage -> Collection Options Sets in the Code Composer Studio™ online help document. In this example, we use the three CST defaults: MinimumSize, Speed, and MaximumSpeed.

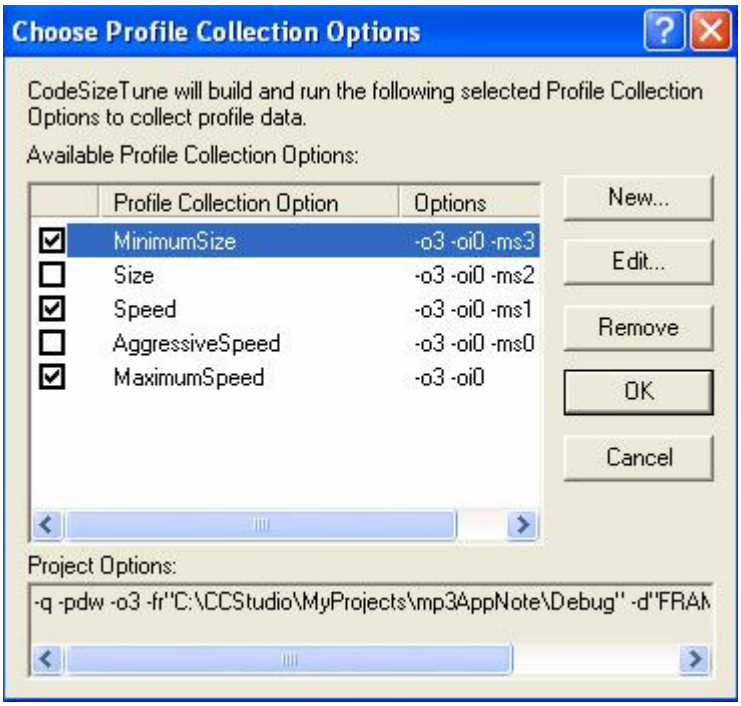

Figure 13. CST Default Collection Options

Select the toolbar button RebuildAll and ReprofileAll (see Figure 11). CST now starts building and profiling the application with different sets of build options (see Figure 13). If you have a large application, this step may take a long time. This MP3 example takes several minutes. The time varies depending on the speed of your computer. When CST is building and profiling the application, you see the progress in the CodeSizeTune tab of the output window as shown in Figure 14. The output shows you which collection options set is currently running.

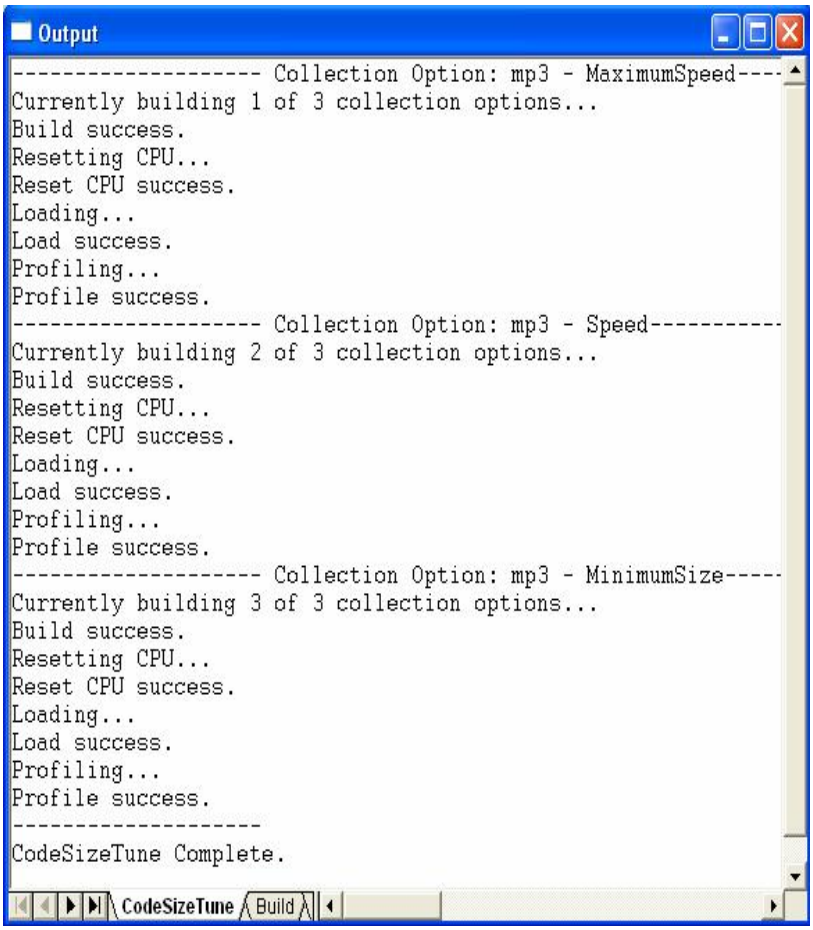

Figure 14. The Output Window

4.4.3 Viewing Possible Solutions and Choosing the Solution Point

When CST finishes building and profiling the application, a graph displays. The series of points on the graph represent the possible tradeoffs between code size and cycle counts. Each point represents your application built with a particular set of options, thus, a possible tradeoff. CST calls each point a “function-specific options set” because each point now corresponds to the function-level build options that CST used to create an application with that code size and performance. If you hold your mouse pointer over a point on the graph, a tool tip with the total code size and total cycle count displays. If you click on a point, the profile viewer window with a two-dimensional table is launched automatically (see Figure 16). This table shows you the corresponding function level options, cycle count, and code size that point represents.

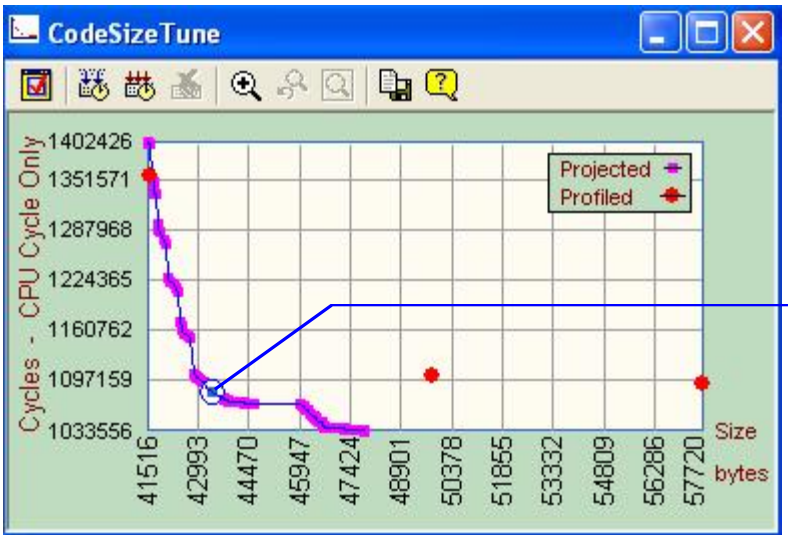

Figure 15. The CST Window After Building and Profiling

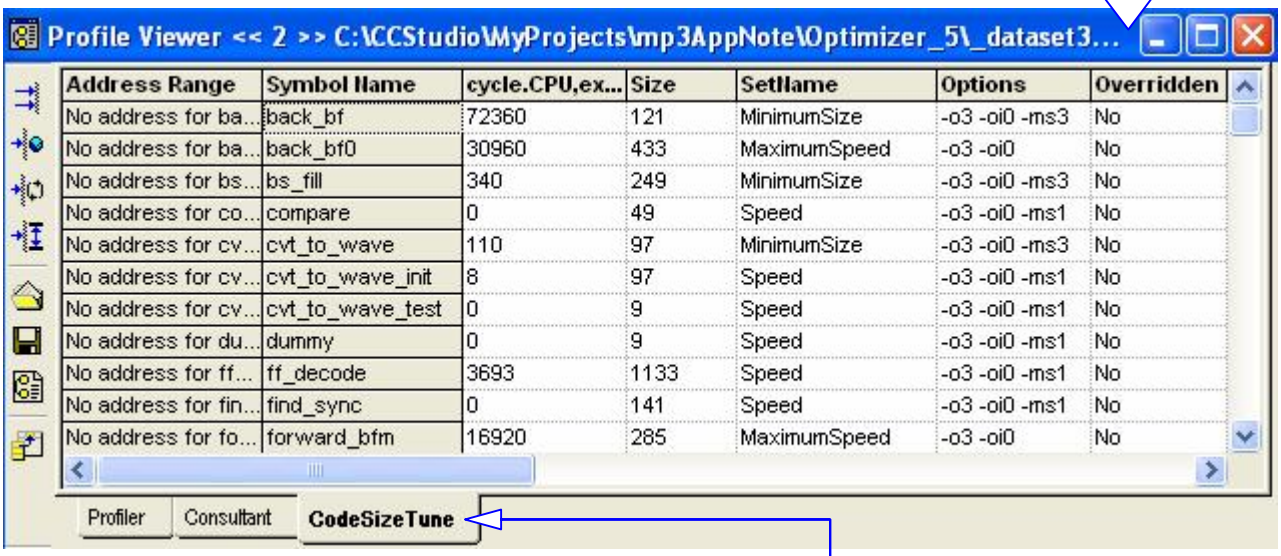

Figure 16. The Profile Viewer Showing the Corresponding Profile Data of the Selected Point

Select the CodeSizeTune tab to see CST-specific data.

The red points in the CST graph are the actual profiled points (see Figure 15). They represent the total code size and cycle count results of the selected collection options sets in the Choose Profile Collection Options dialog. CST records the function-level profile results after rebuilding the project with the specified build options and running the corresponding programs. The total code size and cycle count results are the summation of the function-level profile results. The purple points in the CST graph are projected points. They are computed by CST using the actual profiled points' function-level profile results without rebuilding the project and running the program. Thus, the cycle count and code size results are estimates. To get the actual profile results of a projected point, save the selected projected point to a Code Composer Studio™ IDE project configuration, rebuild the project, and run the program, as shown later in this example.

In some cases, points are close to each other. To more easily differentiate between the points, zoom in with the toolbar button selection Toggle zoom, and then left-click to drag a border around the area of the curve you would like to examine more closely. Deselect the Toggle zoom button to return the cursor to its normal appearance. Change any graph properties by right-clicking on the graph and selecting the menu item Graph Properties….

It is now time to choose the desired solution point. The bottom left-hand corner points on the graph represent the smaller code size and smaller cycle-count, function-specific options sets. Those are the best tradeoffs computed by CST. If you have a code size constraint, select the point that yields the smallest cycle count within the code size constraint. For example, if the code size of your application must be less than 43,400, choose the point that is less than 43,400 on the *x*-axis and lowest on the *y*-axis.

Use the Autopick feature to allow CST to select the best point for you. Right click on the graph, and select the menu item Autopick…. Enter the value of the cycle count or code size in the dialog; CST selects the best point for you.

In this example, the point chosen yields 1,084,165 cycles and 43,388 in code size. Your cycle count and code size result may vary.

4.4.4 Saving Selected Solution Point to a Code Composer Studio™ IDE Project Configuration

In order to rebuild the application and get the actual optimized code size and cycle count (performance), save the selected solution point to a Code Composer Studio™ IDE project configuration. To save a selected point to the configuration, select a point on the graph, and then click the toolbar button Save.

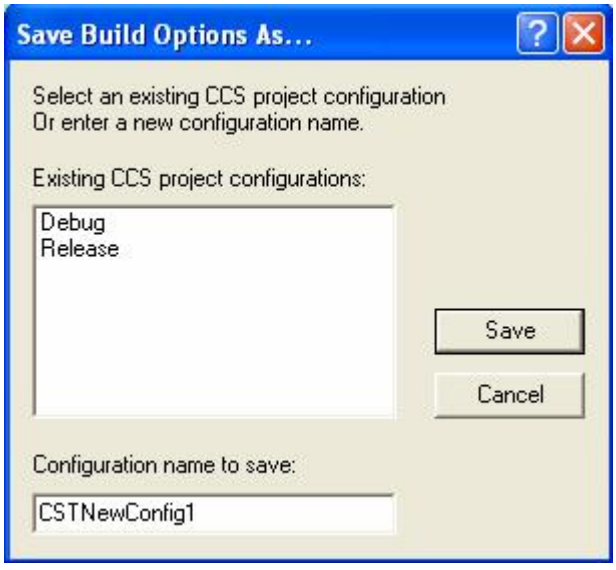

Figure 17. The Save Build Options As… Dialog

The Save Build Options As… dialog then displays (see Figure 17). Enter the configuration name to be saved in the save dialog, and click the OK button to save and close the dialog. Repeat these steps if you have more points to save as different configurations. In this example, we have saved the selected point to the new Code Composer Studio™ IDE configuration, CSTConfig_43388.

You have now completed using CST to tune the code-size of your application. Close the CST window by clicking on the standard close window button on the top right-hand corner of the window. The last saved Code Composer Studio™ IDE configuration is now the active configuration in the project. To see the actual profile data including the function-level code size and cycle count for this active configuration in the profile viewer window, you must rebuild your application, reload, and rerun the program through the Code Composer Studio™ IDE main menu.

4.5 Comparing the Total Code Size and Cycle Count Results After Running CST

Now, compare the code size and cycle count of the newly saved configuration with the original configuration, Debug. The point chosen in the CST graph is a projected point. A projected point is computed by CST without running the program. Thus, the cycle count and code size results are estimates. The estimated results show that the code size improves 25% while not sacrificing any performance.

Next, compare the actual code size and cycle count of the saved configuration, CSTConfig_43388, with the original configuration, Debug, by running the program. If CSTConfig_43388 is not the active configuration, make it active with the Code Composer Studio™ IDE main menu selection Project -> Configurations. To get the actual code size and cycle count, we now build the application with the new configuration using the main menu selection Project -> Rebuild All. Then, load the program with the main menu selection File -> Load Program…, and run the program with the main menu selection Debug -> Run. In this example, load the program *MyProjects\mp3\CSTConfig_43388\mp3.out*.

When the run is complete, the profile viewer updates itself to display the new profile data. Open the profile viewer window if it is not open. Select the CodeSizeTune tab. Add up all the numbers under the *cycle.CPU:Excl.Total* column of the Profiler tab. The actual total cycle count in this example is 1084201. Your cycle count may vary. Now, read the actual code size by opening the map file. In this example, open the *MyProjects\mp3\CSTConfig_43388\mp3.map*. The total size is shown in the first row under the length column of the *.text* section (see Figure 18). The actual total size in this example is 0x0000a980, or 43392 in decimal. Your result may vary. The actual results show that the code size improves 25% while keeping the same performance. Notice that the CST estimates are almost the same as the actual profile results.

```
output                                  attributes/
section   page    origin      length       input sections
--------  ----  ----------  ----------   ----------------
....
.text       0    0000c3a0    0000a980
                 0000c3a0    00004b60     iup.obj (.text)
                 00010f00    00002200     iwinm.obj (.text)
                 00013100    00001a20     isbt.obj (.text)
                 00014b20    00001140     icdct.obj (.text)
                 00015c60    00000880     mhead.obj (.text)
                 000164e0    00000760     towave.obj (.text)
                 00016c40    000000e0     wavep.obj (.text)
```

Figure 18. MP3 Map File Showing the Actual Total Code Size

Furthermore, notice from the CST graph that the more performance degrades, the more code size decreases. In this MP3 project, if you choose other CST projected points, you may improve code size 27% by degrading 11% in performance, or improve code size 17% while increasing performance by 5%.

5 Advanced Features

5.1 Using Overrides Within CST

Using the overrides functionality in CST is optional. In some cases, you may want to directly determine which options are assigned to a particular function or determine that some functions must generate the best code size or the best cycle count rather than let CST select based on its calculations. You can override the collection options set that CST assigns to a function using the Overrides window. Open the Overrides window with the CST toolbar button selection Advanced Options -> View/Apply Overrides.

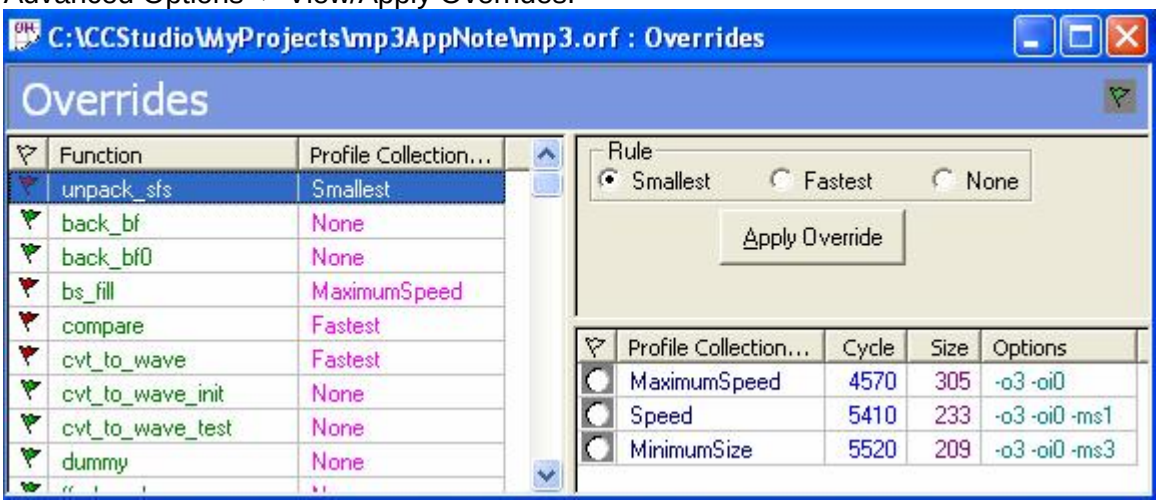

Figure 19. The Overrides Window

When the Apply Override button is selected, the overrides are then applied to all of the option sets that CST projects. The projected points in the CST graph (the purple points in Figure 15) are updated. You can view the overrides information in the Overrides window (as shown in Figure 19) or by selecting the CodeSizeTune tab in the Profile Viewer window (see Figure 16).

The override operation does not apply to the actual profiled points (the red points in the CST graph in Figure 15). If CST reruns, the overrides apply to the projected points but not the actual profiled points.

5.2 Viewing/Modifying Function Level Options After Using CST

After you save a CST point to a new configuration, the new configuration then contains the function-level options that generate the desired code size and cycle counts of the application. This section describes how, as an optional step, to view or modify the function-level options outside of CST.

To view or modify the function-level options, choose the configuration of interest with the Code Composer Studio™ IDE main menu selection Project -> Configurations. You may want to save a copy of this configuration in case you do not like the results after modifying the function-level options. It will save you the time required to rerun CST again. To save the configuration, as an optional step, use the main menu item selection Project -> Project Configurations -> Add….

To launch the Function Level Options dialog, select the menu item Project -> Function Level Options… The dialog displays as shown in Figure 20.

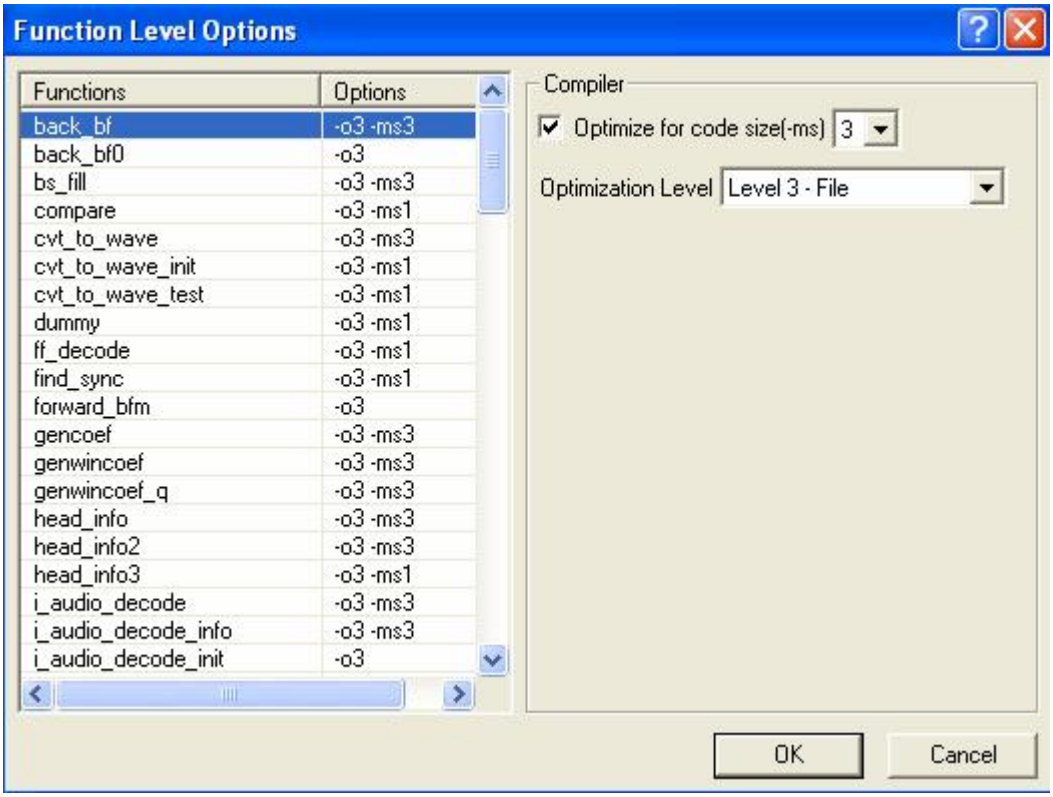

Figure 20. Function Level Options Dialog

To modify the build options, select the function to be modified in the left-hand pane (see Figure 20). Change the compiler options in the right-hand pane. Repeat this step for other functions. Click the OK button to save and close the dialog.

6 Limitation

CST does not allow the user to vary inlining functions. Inlining is an effective method of trading size for speed, particularly if the aggressiveness of inlining can be varied within the application. It therefore provides the exact leverage that CST wishes to exploit. But CST is unable to use automatic inlining because it is unable to vary inlining. Inconsistent inlining between CST collection options sets presents a problem to CST because it now is comparing functions that perform different amounts of work or are being called a different number of times. This invalid comparison leads the CST calculation to choose versions of functions that do not inline and choose versions of inlined functions that are inlined most aggressively. Not only are these two choices inconsistent, they are usually non-optimal even if forced to be consistent.

CST automatically adds the compiler option –oi0 (turn off automatic inlining) to all collection options sets before building and profiling the application. When a solution point is saved to a Code Composer Studio™ IDE configuration, the –oi0 that is added by CST is removed automatically.

7 Conclusion

In conclusion, CST helps developers find the best tradeoff between code size and performance of their application. After executing the application under CST, CST gathers all the relevant profile data and plots it on a graph. The series of points on the graph can be queried interactively. Such interactivity makes finding the solution point quick and easy. The new tuning environment, including the step-by-step instructions in the advice window, provides guidance on using CST. Furthermore, users gain the benefit of viewing and modifying the build options on a per-function basis after using CST. This example illustrates how to reduce code size from 57,568 bytes to 43,392 bytes, a 25% improvement in code size, without losing any performance. In addition, it demonstrates how, as an advanced feature, to use the overrides functionality to override the collection options set that CST assigns to each function and how to modify the build options on a per-function basis after using CST.

Appendix A. Appendix A. What's New and Changed: Code Composer Studio™ IDE v2.2 PBC vs. v3.0 CST

A.1 Stand-alone Code Composer Studio™ IDE Plug-in vs. an Integrated Tuning Tool

In Code Composer Studio™ IDE v2.2, Profile Based Compilation (PBC) is a stand-alone plug-in. It has its own PBC main menu. Users may find it difficult to understand why and when to use PBC. Furthermore, the name PBC does not indicate what this plug-in does.

In Code Composer Studio™ IDE v3.0, the name has changed from PBC to CodeSizeTune (CST) so that users can understand that this plug-in tunes the code size for their application. The new tuning environment allows users to follow the instructions in the advice window and understand why and when to use CST. CST is now available with the menu selection Profile -> Tuning -> CodeSizeTune.

A.2 Graphical User Interface (GUI) Improvements

A.2.1 Old Assistant Window vs. New Advice Window

The advice window of the new tuning suite replaces the old assistant window. All the instructions about how to use CST are now under the CodeSizeTune tab of the advice window. The advantage of using the advice window vs. the assistant window is that it not only gives you the CST step-by-step instructions, but it also gives you overall advice for your application under the General tab. This further advice may lead to additional benefits in using other tuning tools.

A.2.2 Old Assistant Links vs. New Toolbar Buttons

The functional links in the old assistant window now become toolbar buttons at the top of the CST window. Users can focus on just one window (the CST window) instead of two (the assistant window and the graph window) when performing CST operations.

Under PBC, the graph window may or may not be displayed. Under CST, the CST window is always open as long as user is using CST. The window becomes the central view of the tool.

A.2.3 Old PBC Menu Items vs. New Toolbar Buttons

Many frequently used CST operations are moved from the two PBC menus to the CST window toolbar. Under PBC, some operations are accessed through the main PBC menu, and some are accessed through the graph context menu. Users must right-click on the PBC graph to access the context menu items. Now users can easily click on one of the toolbar buttons in front of them to perform frequently used operations such as building, profiling, toggling zoom, undoing zoom, resetting the graph, saving a point, and launching help. The PBC menu and its submenu items are removed. The CST menu item is now under the Code Composer Studio™ IDE main menu Profile ->Tuning without any submenu item.

The advanced options are now grouped into the advanced options dialog. Users clearly understand that these operations are advanced options, and the dialog can be easily launched through the advanced options toolbar button. Examples of the advanced options include changing the CST predefined collection options sets and overriding the function-level options set.

A.2.4 Simplifying the Profile Control Points Setup

PBC profile points, including the exit point, start point, and end point, allow users to tell the profiler where to start and stop collecting profile data. In the new CST, all profile-related points are managed in the profile setup window, not in the PBC menu or the source-editor PBC context menu. These profile points are called exit point, resume point, and halt point in the new profile setup window.

A.2.5 Other Changes

CST has improved the status and error messages in the CST output window. For example, the status message while CST is building and profiling the application for the user has been improved. It now shows the number of the current round of collection options set and the total number of rounds. For example, CST shows “Currently building 1 of 3 collection options…” This way, users can better estimate the time it may take CST to finish all the rounds.

The new CST not only allows users to save the CST projected points (the possible tradeoffs) on the graph, but also to save the profiled points (the CST predefined collection options sets) to different Code Composer Studio™ IDE configurations.

A.3 Profile Results

Profile results of the code size and cycle counts collected under older revisions of Code Composer Studio™ IDE are not exactly the same as before. This difference is due to improvements in the compiler and the simulators.

IMPORTANT NOTICE

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

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

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

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

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

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

Following are URLs where you can obtain information on other Texas Instruments products and application solutions:

Products		**Applications**	
Amplifiers	amplifier.ti.com	Audio	www.ti.com/audio
Data Converters	dataconverter.ti.com	Automotive	www.ti.com/automotive
DSP	dsp.ti.com	Broadband	www.ti.com/broadband
Interface	interface.ti.com	Digital Control	www.ti.com/digitalcontrol
Logic	logic.ti.com	Military	www.ti.com/military
Power Mgmt	power.ti.com	Optical Networking	www.ti.com/opticalnetwork
Microcontrollers	microcontroller.ti.com	Security	www.ti.com/security
		Telephony	www.ti.com/telephony
		Video & Imaging	www.ti.com/video
		Wireless	www.ti.com/wireless

Mailing Address: Texas Instruments
Post Office Box 655303 Dallas, Texas 75265