

## Getting Started With the Code Composer Studio™ Tutorial

The Code Composer Studio™ Tutorial contains lessons that help you get started quickly with the program. If you are a new user, try *Developing a Simple Program* in the IDE module of the tutorial.

Each lesson contains an overview section that provides a summary of the lesson, the learning objectives, the tutorial example used, the application objective, and any prerequisites or special instructions required to successfully complete the lesson.

You can start at the beginning and navigate through the complete tutorial using the navigation links at the bottom of each topic, or you can use the list below to jump start to a module containing lessons you are interested in. Use the print process detailed in the [How to Print](#) topic to print the topics for review.

For demonstrations of the different products and tools, please see the following website for more information: <http://www.ti.com/ccstudiodemos>.

The Code Composer Studio™ Tutorial contains the following modules:

- [Code Composer Studio™ IDE](#)
- [Application Code Tuning](#)
- [Advanced Event Triggering](#)
- [Optimizing C](#)
- [Using DSP/BIOS](#)
- [Using Real-Time Data Exchange](#)
- [Real Time Emulation](#)

To see a list of the examples used in the tutorials for all ISAs, please see the [Tutorial Example List](#).

3/29/2005 13:46

### How to Print

1. Select an Overview or Introduction topic from the Contents tab.
2. Right-click in the topic window and choose Print.
3. In the Print dialog, put a checkmark in the Print all linked documents box (under the Options tab).

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007

## Getting Started With the Code Composer Studio™ Tutorial

Page 2 sur 548

4. Make sure the Print range is set to print all of the pages.
5. Select the printer you want to use. If you want to set any printer properties, click Properties.
6. Click OK.

This action will print the lesson associated with the Overview topic you selected.

You may receive a print error dialog. You may receive this error more than once while printing a lesson. This error can be safely ignored. Simply click OK to close the error dialog.

**Note:** You can find an Overview or Introduction topic by expanding the books displayed in the Contents tab.

### Tutorial Example List

Section	Topic 1	Topic 2	Example
Code Composer IDE	Developing a Simple Program Project Management		Volume 1 mainapplication maximilibrary datdisplay mainapplication
	Editing Techniques		maximilibrary sinewave modern modern modern gelsolid none
	Using Debug Tools Data Visualization Profiling Code Execution Using GEL Language Configuring Target Devices Configuring Simulated Peripherals (28x/24x only)	In & Out Configuration Peripheral Interrupt Timers Peripheral VBUS	inout pie timer vbus
Application Code Tuning	Tools/Tuning Dashboard	Profile Setup Goals Window Advice Window Profile Viewer	modern modern modern modern
	Tools/CodesizeTune Tool Tutorial	Tuning an Application 55x Tuning an Application 6x	mat_oprn mat_oprn_1 modern
		Quick Start Tutorial	modern

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007

Advanced Event Triggering	Tools/Consultant Tutorial Tools/CacheTune Tutorial 55 Tools/CacheTune Tutorial 6x	In-Depth Tutorial	zlib consultant mat_oprn mat_oprn_1
Optimizing C			optimizing_c linear_asm
DSP/BIOS Tutorial	Getting Started	Creating a DSP/BIOS Program	hello 1 hello2 volume2 volume2 volume4 hostio1 hostio2
	Scheduling Program Threads	Debugging Program Behavior Analyzing Real-Time Schedule Connecting to Virtual I/O Devices	echo tsktest
	Sharing and Synchronizing Resource Access	Scheduling SW1 and PRD Threads Using Tasks for Blocking	semtest tsktest
	Managing Memory Handling Input/Output	Using Semaphores to Send Using Semaphores for Mutual Using Mailboxes	mutex mbxtest memtest slotest
RTDX	RTDX Configuration Considerations	Recording RTDX Data in a Log File	S314
	Using RTDX From Target Application	Sending an Integer to the Host Receiving an Integer from the Host Reconfiguring the Target Buffer Sending an Array of Integers to the Host Receiving an Integer from the Host Asynchronously Receiving an Array of Integers from the Host	S111 S113 S113 S112 S114 S115
	Writing RTDX Host Clients	Sending an Integer to the Target Receiving an Integer from the Target Remote Enabling and Disabling of a Channel Receiving Entire Messages Using SAFARRAYs Sending an Array of Integers Using SAFARRAYs	S214 S212 S219 S213 S215

file:///C:/Documents and Settings/Pierre-Armand/Local Settings/Temp/~hhBA53.htm

26/09/2007

Getting Started With the Code Composer Studio Tutorial

Page 4 sur 548

Real-Time Emulation	Advanced RTDX Topics	Receiving Data from Multiple Channels Receiving Data from the ALL Channel Seeking Through Messages Flushing a Channel Sending Structured Data Communicating in a Multiprocessor Environment	S216 S217 S218 S412 S413 S416
	Real-Time Usage	Configuring the Project and Interrupts Debug Mask Bit Usage	realtime dbgmn

## CCStudio IDE Introduction

### CCS - Intro

This tutorial module introduces basic and advanced concepts of the Code Composer Studio IDE. The first lesson, Developing a Simple Program, shows you the basics of putting together a project and how to use common features like build options, breakpoints, Probe Points, watch windows, and graphs. Lessons that follow introduce more advanced topics such as Data Visualization, Using GEL Language, and Profiling Code Execution.

**Note:** If you installed Code Composer Studio in a folder other than C:\CCStudio\_v3.10, use that location wherever this tutorial mentions the C:\CCStudio\_v3.10 location. Some files may have to be loaded manually if the install directory has been changed.

This tutorial module contains the following lessons:

- Developing a Simple Program
- Project Management
- Editing Techniques
- Using Debug Tools
- Data Visualization
- Profiling Code Execution
- Using GEL Language
- Configuring Target Devices



file:///C:/Documents and Settings/Pierre-Armand/Local Settings/Temp/~hhBA53.htm

26/09/2007

## Developing a Simple Program

CCS - L1

In this lesson, you develop a simple program that performs basic signal processing.

Learning Objectives:

- Create a simple program using Code Composer Studio™ tools
- Use basic debug techniques to analyze the program
- Use the tools to facilitate development of programs

### **Examples used in this lesson: volume1**

Application Objective:

This lesson uses the `volume1` example to develop and run a simple program. You will create a project from scratch, add files to it, and review the code. After you build and run the program, you will change build options using the build options dialog, and fix syntax errors using the editor. Finally, you will utilize basic debug techniques like breakpoints, watch windows and file I/O.

This lesson consists of the following steps. To go directly to a step, click on the link below:

- [Creating a New Project](#)
- [Adding Files to a Project](#)
- [Reviewing the Source Code](#)
- [Building and Running the Program](#)
- [Changing Program Options and Fixing Syntax Errors](#)
- [Using Breakpoints and the Watch Window](#)
- [Using the Watch Window With Structures](#)
- [Adding a Probe Point for File I/O](#)
- [Displaying Graphs](#)
- [Animating the Program and Graphs](#)

[file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm](#)

26/09/2007

## Getting Started With the Code Composer Studio Tutorial

Page 6 sur 548

[Adjusting the Gain](#)



The forward arrow will take you to the next page in this lesson.

### **Creating a New Project**

CCS - L1

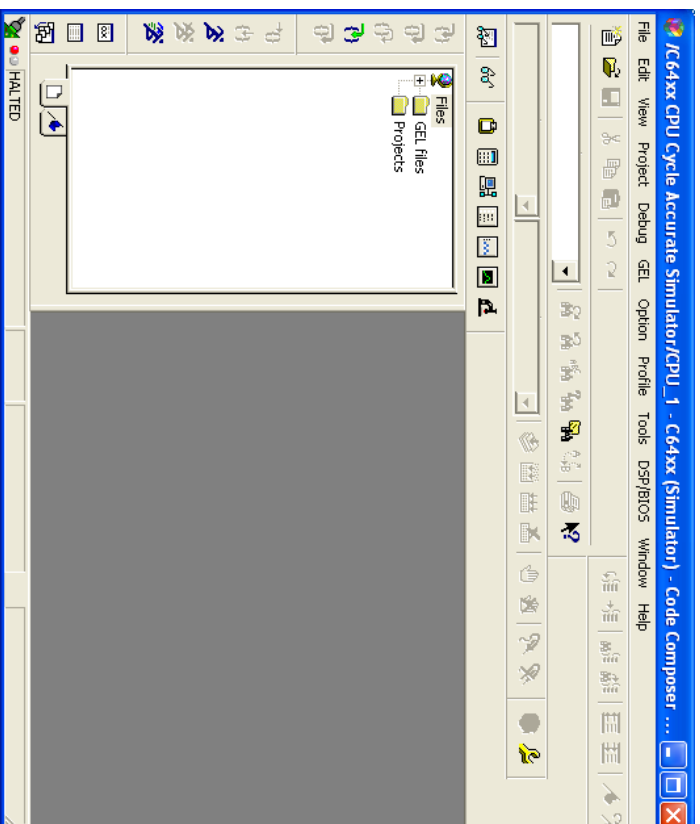
In this lesson, you create a project with the Code Composer Studio™ IDE and add source code files and libraries to the project. Over the course of the lesson, you will learn about the following types of files:

- `.lib` This library provides runtime support for the target DSP chip
- `.c` This file contains source code that provides the main functionality of this project
- `.h` This file declares the buffer C-structure as well as define any required constants
- `.pj1` This file contains all of your project build and configuration options
- `.asm` This file contains assembly instructions
- `.cmd` This file maps sections to memory

1. If you installed the Code Composer Studio™ program in `c:\CCStudio_v3.1.0`, create a folder called `volume1` in the `C:\CCStudio_v3.1.0\myprojects` folder. (If you installed elsewhere, create a folder within the `myprojects` folder in the location where you installed.)
2. Copy the contents of `C:\CCStudio_v3.1.0\tutorial\target\volume1` folder to this new folder.
3. If you have not already done so, from the Windows Start menu, choose Programs→Texas Instruments→Code Composer Studio 3.1→Code Composer Studio 3.1. (Or, double-click the Code Composer Studio icon on your desktop.)

[file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm](#)

26/09/2007



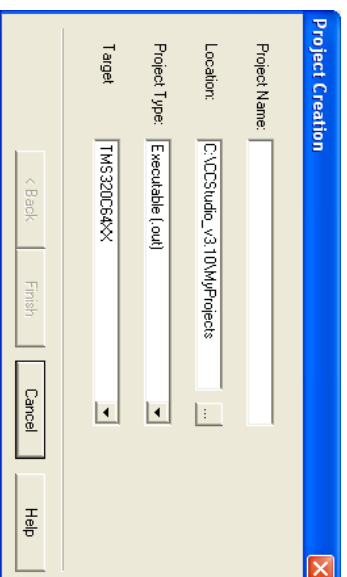
4. From the Project menu, choose New.

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007

Getting Started With the Code Composer Studio Tutorial

Page 8 sur 548



5. In the Project Name field, type volume1.
6. In the location field, browse to the working folder you created in step 1.
7. In the Project Type field, select Executable (.out).
8. In the Target field, select your target configuration and click Finish.  
The Code Composer Studio™ Program creates a project file called volume1.pjt. This file stores your project settings and references the various files used by your project.



### Adding Files to a Project

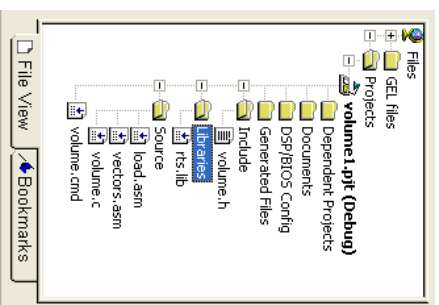
CCS - LI

1. Choose Project→Add Files to Project. Select volume.c from the working folder you created, and click Open. You can also add files to the project by right-clicking on the Project View icon and choosing Add Files to Project or by dragging and dropping files into folders in the Project View window.
2. Choose Project→Add Files to Project. Select Asm Source Files (\*.a\*,\*.s\*) in the Files of type box. Select vectors.asm and load.asm, and click Open. Asm files contain assembly instructions needed to set the RESET interrupt to branch to the program's C entry point, C\_int00. (For more complex programs, you can define additional interrupt vectors in vectors.asm, or you can use DSP/BIOS to define all the interrupt vectors automatically.)
3. Choose Project→Add Files to Project. Select Linker Command File (\*.cmd,\*.lcf) in the Files of type box. Select volume.cmd and click Open. This file maps sections to memory.

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007

4. Choose Project→Add Files to Project. Go to the compiler library folder (C:\CCStudio\_v3.10\c6000\c6000\lib). Select Object and Library Files (\*.\*.o\*, \*.J) in the Files of type box. Select the `rts.lib` file for the `target` you are configured for and click Open. This library provides run-time support for the target DSP. For some targets, the runtime library may have a more specific file name, for example, `rts6200.lib`.
5. In the Project View window, right-click on `volume1.pjt` and select Scan All File Dependencies. Click on the `volume1` project in the Project View window to view `volume.h` under the Include folder.
6. Expand the Project list by clicking the + signs next to Projects, `volume1.pjt`, Libraries, and Source. This list is called the Project View. This is a generic view of the project, your `.lib` file may differ.



**Note:** If you do not see the Project View, choose View→Project. Click the File icon at the bottom of the Project View if the Bookmarks icon is selected.

You do not need to manually add include files to your project, because the program finds them automatically when it scans for dependencies as part of the build process. After you build your project, the include files appear in the Project View.

If you need to remove a file from the project, right click on the file in the Project View and choose Remove from Project in the pop-up menu.

When building the program, the program finds files by searching for project files in the following path order:

- The folder that contains the source file.

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007

## Getting Started With the Code Composer Studio Tutorial

Page 10 sur 548

- The folders listed in the Include Search Path for the compiler or assembler options (from left to right).

### Reviewing the Source Code

CCS - LI

Double-click on the `volume.c` file in the Project View to open the source code in the right half of the Code Composer Studio™ window.

Note the following functions within the code:

- After the main function prints a message, it enters an infinite loop. Within this loop, it calls the `dataIO` and processing functions.
- The processing function multiplies each value in the input buffer by the gain and puts the resulting values into the output buffer. It also calls the assembly load routine, which consumes instruction cycles based on the `processingLoad` value passed to the routine.
- The `dataIO` function in this example does not perform any actions other than to return. Rather than using C code to perform I/O, we will use a Probe Point to read data from a file on the host into the `inp_buffer` location.

[volume.c](#) (Click to view code)


### Building and Running the Program

CCS - LI

The program automatically saves changes to the project setup as you make them. In case you exited after the previous section, you can return to your stopping point by restarting Code Composer Studio™ IDE and using Project→Open.

**Note:** If the program displays an error message saying it cannot initialize the target DSP, choose the Debug→Reset CPU menu item, then File→Reload Program. If this does not correct the problem, you may need to run a reset utility provided with your target board.



To build and run the program, follow these steps:

1. Choose Project→Rebuild All or click the  (Rebuild All) toolbar button. The program recompiles, reassembles, and relinks all the files in the project. The Build frame at the bottom of the window displays messages about this process.
2. By default, the `.out` file is built into a debug directory located under your current project folder. You can change this location by selecting a different one from the tool bar.

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007



3. Choose File→Load Program. Select the program you just rebuilt, Volume1.out, and click Open. (It should be in the C:\CCStudio\_v3.10\myprojects\volume1\Debug folder unless you set up your project directory elsewhere.) CCSStudio loads the program onto the target DSP and opens a Disassembly window that shows the disassembled instructions that make up the program. (A Stdout tab may open at the bottom of the window to show any output the program sends to stdout.)
4. In the displayed disassembly window, click on an assembly instruction in the mixed-mode window. (Click on the actual instruction, not the address of the instruction or the fields passed to the instruction.) Press the F1 key to search for help on the disassembled instruction. This is a good way to get help on an unfamiliar instruction.
5. Choose Debug→Go Main to begin execution from the main function. The execution halts at main and is identified by .
6. Choose Debug→Run, or click the  (Run) toolbar button. The text "Volume example started" should appear in the Stdout tab of the message window on the bottom of the CCSStudio™ screen.
8. Choose Debug→Halt to quit running the program.
9. From the View menu, choose Mixed Source/ASM to un-check the menu item if it is checked on. This allows you to view c code without the assembly so you can accomplish the next task: Changing Program Options and Fixing Syntax errors.



### Changing Program Options and Fixing Syntax Errors

CCS - LI

In the previous section, the portion of the program enclosed by the preprocessor commands (#ifdef and #endif) did not run because FILEIO was undefined. In this section, you set a preprocessor option. You also find and correct a syntax error.

1. Choose Project→Build Options.
2. In the Compiler tab of the Build Options window, select Preprocessor from the Category list. Type FILEIO in the Pre-Define Symbol [-D] field. Press the Tab key.

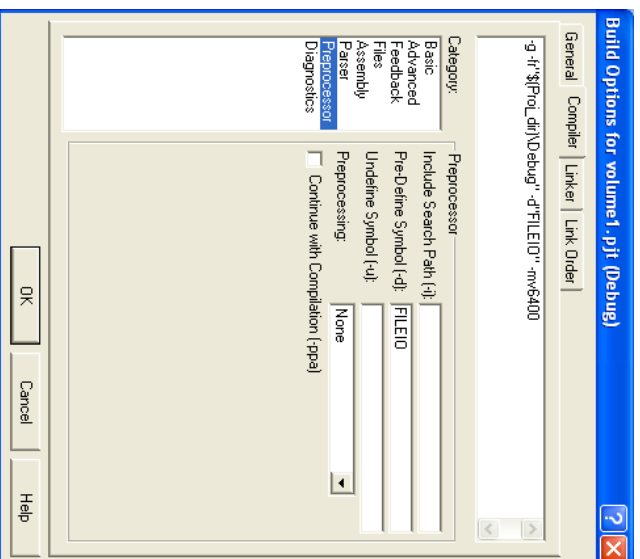
Notice that the compiler command at the top of the window now includes the -d option. The code after the #ifdef FILEIO statement in the program is now included when you recompile the program. (The other options may vary depending on the DSP board you are using.)


file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007

Getting Started With the Code Composer Studio Tutorial

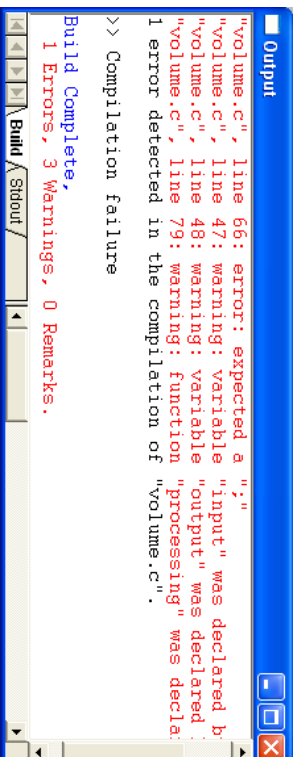
Page 12 sur 548



3. Click OK to save your new option settings.
4. Choose Project→Rebuild All or click the  (Rebuild All) toolbar button. You need to rebuild all the files whenever the project options change.
5. A build message indicates that the program contains compile errors. Click the Build tab and scroll up in the Build tab area. You will see a syntax error message. Your line numbers and ISA reference may vary.

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007



```

"Volume.c", line 66: error: expected a ";"
"Volume.c", line 47: warning: variable "input" was declared by
"Volume.c", line 48: warning: variable "output" was declared by
"Volume.c", line 79: warning: function "processing" was declar
1 error detected in the compilation of "Volume.c".
>> Compilation failure
Build Complete,
1 Errors, 3 Warnings, 0 Remarks.

```

6. Double-click on the line that describes the location of the first syntax error: **expected a ";"**. (actual line number may vary). Notice that the volume.c source file opens, and your cursor should be on the following line:

```
processing(input, output);
```

7. Fix the syntax error in the following line above the cursor location (The semicolon is missing.) It should look like this:

```
puts("begin processing");
```

8. Notice that an asterisk (\*) appears next to the filename in the Edit window's title bar, indicating that the source file has been modified. The asterisk disappears when the file is saved.

9. Choose File→Save or press Ctrl+S to save your changes to volume.c.

10. Choose Project→Build or click the  (Incremental Build) toolbar button to rebuild updated files.

11. Choose File→Load Program and select volume1.out.

**Tip: You can auto-load your program every time you build by customizing your project environment. Choose Option→Customize, and click on the Program/Project Load tab. Then, click on the Load Program After Build option.**

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

13. Choose Debug→Run or click the  (Run) toolbar button. The text "Volume example started", followed by "begin processing" should appear in the Stdout tab of the message window on the bottom of the CCS studio™ screen.

14. Choose Debug→Halt to quit running the program.

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007

Getting Started With the Code Composer Studio Tutorial

Page 14 sur 548



### Using Breakpoints and the Watch Window

CCS - 11


When you are developing and testing programs, you often need to check the value of a variable during program execution. In this section, you use breakpoints and the Watch Window to view such values. You also use the step commands after reaching the breakpoint. There are both software and hardware breakpoints, but this tutorial only uses software breakpoints. Hardware breakpoints are implemented by the hardware internally, and can be set in any memory type. They are typically used when software breakpoints will not work, such as in memory that cannot be written to, like ROM memory. Software breakpoints are implemented as an opcode replacement, and there is no limit to the number of software breakpoints that can be set.

1. Choose File→Reload Program.

2. Double-click on the volume.c file in the Project View. You may want to make the window larger so that you can see more of the source code at once.

3. Put your cursor in the line 61 (your line number may vary) under the main function that says:

```
data10();
```

4. Click the  (Toggle Breakpoint) toolbar button or press F9. This will set a software breakpoint. The selection margin indicates that a breakpoint has been set (red icon).

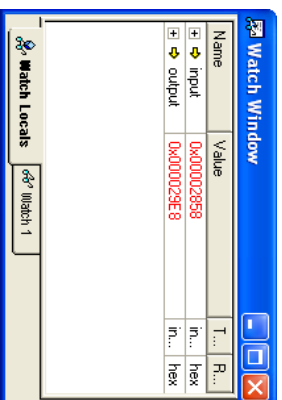
5. Choose View→Watch Window. A separate area in the lower-right corner of the main window appears. At run time, this area shows the values of watched variables. By default, the Watch Locals tab is selected and displays Local variables that are local to the executed function.


6. If not at main, choose Debug→Go Main.

7. Choose Debug→Run, or press F5, or press the  icon. Your numbers may vary.




file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007



8. Select the Watch1 tab.
9. Click on the expression icon  in the Name column and type dataIO as the name of the variable to watch.
10. Click on the white space in the watch window to save the change. The value should immediately appear.



11. Click the  (Step Over) toolbar button or press F10 to step over the call to dataIO().
12. Experiment with the step commands:
  -  Step Into (F8)
  -  Step Over (F10)

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007

Getting Started With the Code Composer Studio Tutorial

Page 16 sur 548


-  Step Out (Shift F7)
  -  Run to Cursor (Ctrl F10)
13. After you have finished experimenting, click  (Remove all breakpoints) before proceeding to the next part of the lesson. This will remove all software and hardware breakpoints.



### Using the Watch Window with Structures

CCS - LI

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

1. Select the Watch1 tab.
2. Click on the expression icon  in the Name column and type str as the name of the expression to watch.
3. Click on the white space in the watch window to save the change. The value should immediately appear as in the following example.



4. Recall from [Reviewing the Source Code](#) that a structure of type PARMS was declared globally and initialized in volume.c. The structure type is defined in volume.h.
5. Click once on the + sign next to str, to expand this line to list all the elements of the structure and their values. (The values may vary.)

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007





You can double-click on the Value of any element in the structure to edit the value for that element.

- In the Value column of the Watch window, change the value of a variable. The value in the Watch Window changes color to red, indicating that you have changed it manually.
- Select the str variable in the Watch Window and hit the Delete key. Repeat this step for all expressions in the Watch Window.
- Choose Debug→Breakpoints. In the Breakpoints tab, if there are any breakpoints left, click Delete All, and then click OK.



### Adding a Probe Point for File I/O

CCS - LI

In this section, you add a Probe Point, which reads data from a file on your PC. Probe Points are a useful tool for algorithm development. You can use them as follows:

- To transfer input data from a file on the host PC to a buffer on the target for use by the algorithm
- To transfer output data from a buffer on the target to a file on the host PC for analysis
- To update a window, such as a graph, with data

[More About Probe Points](#) (Click to view additional information)

This lesson shows you how to use a Probe Point to transfer the contents of a PC file to the target for use as test data. It also uses a breakpoint to update all the open windows when the Probe Point is reached. As with breakpoints, there are both software and hardware probe points, this tutorial will only use software probe points.

- Choose File→Load Program. Select volume1.out, and click Open.

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm


26/09/2007

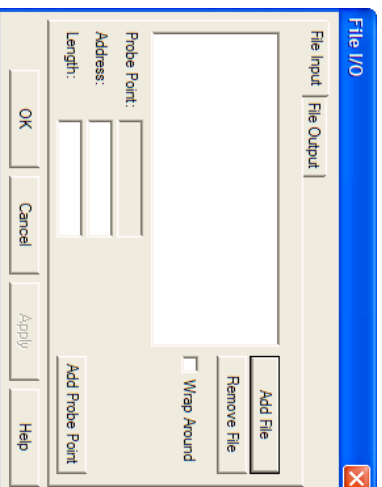
### Getting Started With the Code Composer Studio Tutorial

Page 18 sur 548

- Double-click on the volume.c file in the Project View.
- Put your cursor in line 61 (your line number may vary) of the main function that says: data10();

The data10 function acts as a placeholder, you will add to it later. For now, it is a convenient place to connect a Probe Point that injects data from a PC file.

- Click the  (Toggle Probe Point) toolbar button. The selection margin indicates that a probepoint has been set (blue icon).
- From the File menu, choose File I/O. The File I/O dialog appears, and you may now select input and output files.



- In the File Input tab, click Add File.
- Browse to the volume1 project folder you created, select sine.dat and click Open.

**Note:** Notice that you can select the format of the data in the Files of Type box. The sine.dat file contains hex values for a sine waveform.

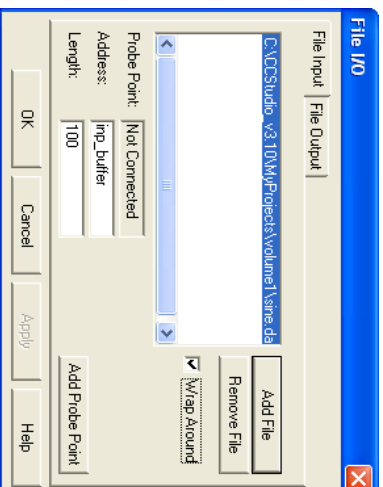
A control window for the sine.dat file appears. (It may be covered by the main window.) Later, when you run the program, you can use this window to start, stop, rewind, or fast forward within the data file.

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007



8. In the File I/O dialog, change the Address to inp\_buffer and the Length to 100. Also, put a check mark in the Wrap Around box.



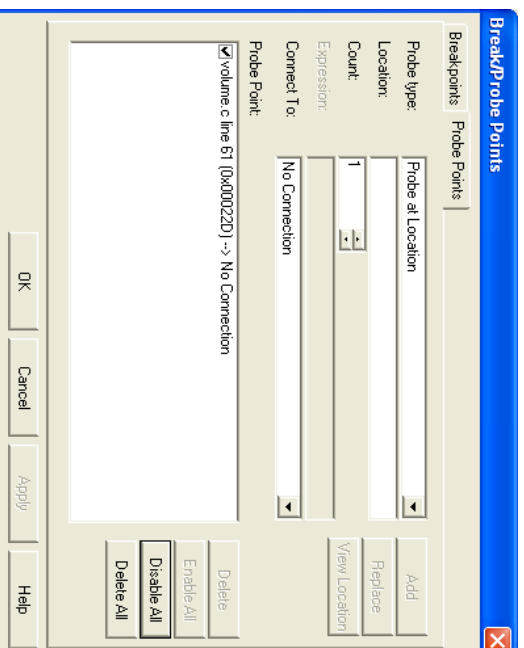
- The Address field specifies where the data from the file is to be placed. The inp\_buffer is declared in volume.c as an integer array of BUFSIZE (a constant that is defined in volume.h).
  - The Length field specifies how many samples from the data file are read each time the Probe Point is reached. You use 100 because that is the value set for the BUFSIZE constant in volume.h (0x64).
  - The Wrap Around option causes the IDE to start reading from the beginning of the file when it reaches the end of the file. This allows the data file to be treated as a continuous stream of data even though it contains only 1000 values and 100 values are read each time the Probe Point is reached.
9. Click Add Probe Point. The Probe Points tab of the Break/Probe Points dialog appears.

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007

Getting Started With the Code Composer Studio Tutorial

Page 20 sur 548



10. In the Probe Point list, highlight the line that says VOLUME.C line 61 --> No Connection. Your line number may vary.
11. In the Connect To field, click the down arrow and select the FILE IN:C:\...\sine.dat file from the list.
12. Click Replace. The Probe Point list changes to show the Probe Point connected to the sine.dat file.
13. Click OK. The File I/O dialog shows that the file is now connected to a Probe Point.
14. Click OK to close the File I/O dialog.



**Displaying Graphs**

If you ran the program now, you would not see much information about what the program was doing. You could set watch variables on addresses within the `inp_buffer` and `out_buffer` arrays, but you would need to watch a lot of variables and the display would be unmanageable rather than visual.

There are a variety of ways to graph data processed by your program. In this example, you view a signal plotted against time. You open the graphs in this section and run the program in the next section.

1. Choose **View**→**Graph**→**Time/Frequency**.
2. In the **Graph Property Dialog**, change the **Graph Title**, **Start Address**, **Acquisition Buffer Size**, **Display Data Size**, **DSP Data Type**, **Autoscale**, and **Maximum Y-value** properties to the values shown here. Scroll down or resize the dialog box to see all the properties.



3. Click **OK**. An **Input graph window** for the **Input Buffer** appears.
4. Right-click on the **Input graph window** and choose **Clear Display** from the pop-up menu.

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007

Getting Started With the Code Composer Studio Tutorial

Page 22 sur 548


5. Choose **View**→**Graph**→**Time/Frequency** again.
6. This time, change the **Graph Title** to **Output** and the **Start Address** to `out_buffer`. All the other settings are correct.
7. Click **OK** to display the **Output graph window**. Right-click on the **Output graph window** and choose **Clear Display** from the pop-up menu.




### Animating the Program and Graphs

CCS - LI

So far, you have placed a **Probe Point**, which temporarily halts the target, transfers data from the host PC to the target, and resumes execution of the target application. However, the **Probe Point** does not cause the graphs to be updated. In this section, you create a **breakpoint** that causes the graphs to be updated and use the **Animate** command to resume execution automatically after the **breakpoint** is reached.

1. In the **volume.c** window, put your cursor in the line that calls `dataIO`.
2. Click the  (**Toggle Breakpoint**) toolbar button or press **F9**. A red icon appears in the selection margin next to the blue icon, to designate the **breakpoint**.

You put the **breakpoint** on the same line as the **Probe Point** forcing the target to halt only once to perform both operations—transferring the data and updating the graphs.

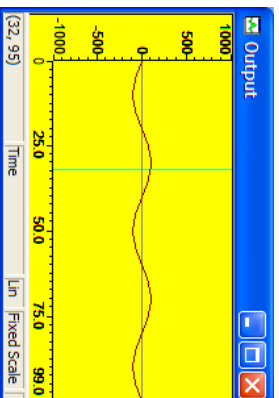
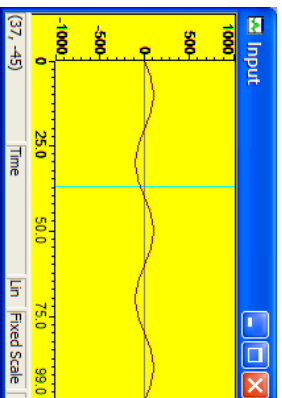
3. Arrange the windows so that you can see both graphs.
4. Click the  (**Animate**) toolbar button or press **F12** to run the program.

[More about the Animate Command](#) (Click to view additional information)

5. Each time the **Probe Point** is reached, the IDE gets 100 values from the `sine.dat` file and writes them to the `inp_buffer` address.

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007



- Choose Debug→Halt to quit running the program.

**Note:** Code Composer Studio™ IDE briefly halts the target whenever it reaches a Probe Point. Therefore, the target application may not meet real-time deadlines if you are using Probe Points. At this stage of development, you are testing the algorithm. Later, you can analyze real-time behavior using RTDX and DSP/BIOS.



### Adjusting the Gain

CCS - 11

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007


### Getting Started With the Code Composer Studio Tutorial

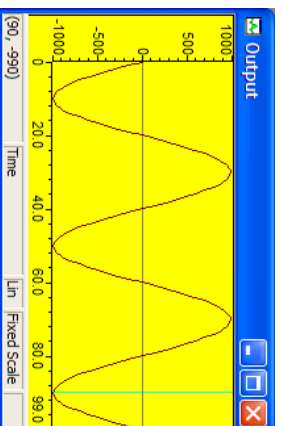
Page 24 sur 548

Recall from [Reviewing the Source Code](#) that the processing function multiplies each value in the input buffer by the gain and puts the resulting values into the output buffer. It does this by performing the following statement within a while loop:

```
*output++ = *input++ * gain;
```

This statement multiplies a value in inp\_ buffer by the gain and places it in the corresponding location in the out\_ buffer. The gain is initially set to MINGAIN, which is defined as 1 in volume.h. To modify the output, you need to change the gain. One method is to use a watch variable.

- Choose View→Watch Window and select the Watch1 tab.
- Click on the expression icon  in the Name column and type gain as the name of the variable to watch.
- Click on the white space in the watch window to save the change. The value should immediately appear.
- If you have halted the program, choose Run from the Debug menu to restart the program. Observe the input and output graph you created earlier.
- From the Debug menu, choose Halt.
- In the Watch Window, select the value of gain (1) and change it to 10.
- From the Debug menu, choose Run. Notice that the amplitude of the signal in the Output graph changes to reflect the increased gain as in the following example:



- From the Debug menu, choose Halt.
- From the Project menu, choose Close.
- Save the program when prompted, and Close all other windows and graphs.

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007

11. Remove all probe points and breakpoints before moving to the next lesson.

This concludes Developing a Simple Program.



## Project Management

---

CCS - L2

In this lesson, you will learn how to use the Advanced Project Management features in the context of building a small application and accompanying library file.

Learning Objectives:

- View the effects of different types of build options within projects
- Create and work with multiple projects
- Use source control methods to integrate projects
- Create library files
- Create and generate makefiles

Examples used in this lesson: myapplication (as part of this lesson, you will create two projects, mainapplication and maxminlibrary, to demonstrate Project Management techniques.)

Application Objective:

This lesson uses the myapplication example to develop some simple programs and use the advanced project management features of the IDE. You will create a library project and an executable application, add files to it, and review the code. After you build and run the programs, you will customize the build options, and use tools including version control and configuration to manage multiple projects. You will also set the link order inside the projects. Finally, you will create and generate an external makefile.

This lesson consists of the following steps. To go directly to a step, click on the link below:

[Creating Files](#)

[Creating a Library Project](#)

[Creating an Executable Application](#)

[Building Files and Projects](#)

[file:///C:/Documents and Settings/Pierre-Armand/Local Settings/Temp/~hhBA53.htm](#)

26/09/2007

## Getting Started With the Code Composer Studio Tutorial

Page 26 sur 548

[Pre and Post Build Customization](#)

[Viewing Multiple Projects](#)

[Using Version Control](#)

[Multiple Project Configurations](#)

[Setting Link Order](#)

[Editing a Project File](#)

[Creating an External Makefile](#)

[Generating an External Makefile](#)



The forward arrow will take you to the next page in this lesson.

## Creating Files

---

CCS - L2

### Creating ANSI C Source Files

In this exercise you will create a project folder and three ANSI C source text files containing two simple functions and one test application. Over the course of the lesson, you will learn about the following types of files:

- .lib This library provides runtime support for the target DSP chip
- .c This file contains source code that provides the main functionality of this project
- .jdt This file contains all of your project build and configuration options

### Creating testapp.c

1. In Windows Explorer, browse to [C:\CCStudio\\_v3.10\myprojects\](#).

[file:///C:/Documents and Settings/Pierre-Armand/Local Settings/Temp/~hhBA53.htm](#)

26/09/2007

2. Create a folder named myapplication.
3. Create two folders inside the myapplication folder, called mainapplication and maxminlibrary respectively.
4. If you have not already done so, start Code Composer Studio™ IDE.
5. From the File menu, select **New->Source File**.
6. Select the following link to view testapp.c: Copy and paste the text into the source file you just created.  
[testapp.c \(Click to view code\)](#)
7. From the File menu, choose Save As.
8. Browse to [C:\CCStudio\\_v3.10\myprojects\myapplication\](#).
9. Type testapp.c as the file name and click Save.

---

#### Creating minimumvalue.c

1. From the File menu, select **New->Source File**.
  2. Select the following link to view the content of minimumvalue.c: Copy and paste the text into the source file you just created.  
[minimumvalue.c \(Click to view code\)](#)
  3. From the File menu, choose Save As.
  4. Browse to [C:\CCStudio\\_v3.10\myprojects\myapplication\](#).
  5. Type minimumvalue.c as the file name and click Save.
- 

#### Creating maximumvalue.c

1. From the File menu, select **New->Source File**.
2. Select the following link to view the content of maximumvalue.c: Copy and paste the text into the source file you just created.

[file:///C:/Documents and Settings/Pierre-Armand/Local Settings/Temp/~hhBA53.htm](#)

26/09/2007

#### Getting Started With the Code Composer Studio Tutorial

Page 28 sur 548

[maximumvalue.c \(Click to view code\)](#)

3. From the File menu, choose Save As.
4. Browse to [C:\CCStudio\\_v3.10\myprojects\myapplication\](#).
5. Type maximumvalue.c as the file name and click Save.



#### Creating a Library Project

---

CCS - L2

In this exercise you will create a library project called maxminlibrary. Later, when you create an executable application, you will use maxminlibrary to find maximum and minimum integer values within an array.

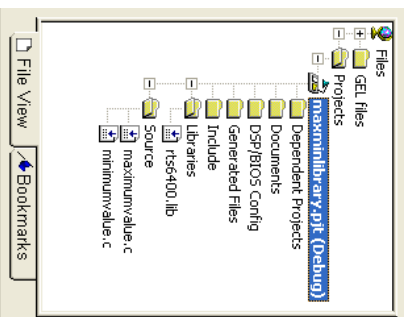
1. From the Project menu, choose New.
2. In the Project Name field, type maxminlibrary as the name of your project.
3. In the location field, type [C:\CCStudio\\_v3.10\myprojects\myapplication\maxminlibrary\](#), or browse to that location.
4. In the Project Type drop-down menu, select Library (.lib) as the project type.
5. In the Target Family drop-down menu, select the target family for your application. For more information about targets, see [Configuring Target Devices](#).
6. Click Finish.
7. In the Project View window, right-click on the project name and click Add Files To Project.
8. From the Files of type drop-down menu, select C Source Files.
9. Browse to [C:\CCStudio\\_v3.10\myprojects\myapplication\](#).
10. Select maximumvalue.c and click Open.
11. Repeat steps 9 and 10 to add minimumvalue.c.
12. In the Project View window, right-click maxminlibrary.lib and select Add Files To Project.

[file:///C:/Documents and Settings/Pierre-Armand/Local Settings/Temp/~hhBA53.htm](#)

26/09/2007

13. From the Files of type drop-down menu, select Object and Library Files.
14. Browse to `C:\CCStudio_v3.10\c6000\cgtools\lib\`, select an `rs.lib` and click Open.

Your Project view should look something like this, although your `rs.lib` files will be target specific:



15. From the Project menu, choose Rebuild All. This should build the file `maxminlibrary.lib`.



### Creating an Executable Application

CCS - 12

Now you will build an executable project that imports and uses the library file created in the previous exercise. In doing so, we will demonstrate how to use various build options.

1. From the Project menu, choose New.
2. In the Project Name field, type `mainapplication` as the name of your project.
3. In the Location field, type `C:\CCStudio_v3.10\myprojects\myapplication\mainapplication\`.

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007

### Getting Started With the Code Composer Studio Tutorial

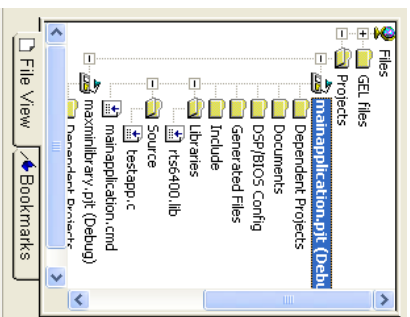
Page 30 sur 548

4. In the Project Type drop-down menu, select Executable (.out) as the project type.
5. In the Target Family drop-down menu, select the target family for your application. For more information about targets, see [Configuring Target Devices](#).
6. Click Finish.
7. In the Project View window, right-click on the project name (`mainapplication.pjt`) and click Add Files to Project.
8. From the Files of type drop-down menu, select C Source Files.
9. Browse to `C:\CCStudio_v3.10\myprojects\myapplication\`.
10. Select `testapp.c` and click Open.
11. In the Project View window, right-click `mainapplication.pjt` and select Add Files to Project.
12. From the Files of type drop-down menu, select Linker Command File.
13. Browse to `C:\CCStudio_v3.10\tutorial\target\maxminmath\`, select `mainapplication.cmd` and click Open.
14. In the Project View window, right-click `mainapplication.pjt` and select Add Files to Project.
15. From the Files of type drop-down menu, select Object and Library Files.
16. Browse to `C:\CCStudio_v3.10\c6000\cgtools\lib\`, select an `rs.lib` and click Open.

Your Project view should look something like this, although your `rs.lib` files will be target specific:

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007



Now, instead of re-compiling the entire file set including the library files, we can utilize Project Level Build and only compile files in mainapplication.pjt.

17. Right-click on mainapplication.pjt and select Build. This will compile only the files within the mainapplication project. Compiling this project produces the following error:

```
>> error: symbol referencing errors - ./Debug/mainapplication.out not built
```

This occurs when the mainapplication project has not defined the library functions being called.

18. Right-click on mainapplication.pjt and select Add Files to Project.
19. Browse to C:\CCStudio\_v3.10\myprojects\myapplication\maxminlibrary\Debug.
20. Select All files or .lib in the drop down list to view the files. Select maxminlibrary.lib and click Open.
21. Right-click on mainapplication.pjt and select Build. Your build should now be successful.



### Building Files and Projects

CCS - L2

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007

### Getting Started With the Code Composer Studio Tutorial

Page 32 sur 548

In this exercise you will learn different methods to build projects and files.

#### Compiling a Particular File

Suppose you wish to verify a particular file's syntax. One method that avoids re-compiling the entire project is the Compile File option. This option allows file specific compilation.

To modify the main application file, testapp.c so that instead of simply computing the maximum and minimum value, it computes the spread of the integer data provided, follow these steps:

1. In the Project View window, double-click on testapp.c to make it the active file.
2. After the line that says: // call library functions, add the following printf function:

```
printf( "The spread of the data is %d\n", max_value - min_value );
```

```
testapp.c
// call library functions
printf( "The spread of the data is %d\n", max_value - min_value )
printf( "The maximum value in the data is %d\n", max_value )
printf( "The minimum value in the data is %d\n", min_value )
return 0;
}
```

3. In the Project View window, right-click on testapp.c and choose Compile File.

**Note:** This feature will not compile the individual file and link it to the pre-existing object files.

#### Excluding a File from Being Compiled

Suppose you would like to exclude a file from being built. This can be accomplished using the Exclude File from Build option. This feature is particularly important when your project code makes use of a soon-to-be-implemented library call/function.

1. From the File menu, select New-Source File.

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007



2. Enter the following text:

```
#include <stdio.h>
int main()
{
    int test_data[] = { 7, 13, 3, 25, 64, 15 };
    // call library functions
    printf( "The average value of the data is %d\n", averageValue(test_data, 5) );
    return 0;
}
```

3. From the File menu, choose Save As.
4. Browse to C:\CCStudio\_v3.10\myprojects\myapplication\.
5. Type anotherestapp.c as the file name and click Save.
6. In the Project View window, right-click on mainapplication.pjt and click Add Files to Project.
7. Browse to C:\CCStudio\_v3.10\myprojects\myapplication\.
8. Select anotherestapp.c and click Open.

Compiling anotherestapp.c would result in a series of errors because it redefines the main() function and also because the averageValue(int [], int) is not defined within the maxminlibrary library file. To allow the rest of the project to compile, you can use the Exclude File from Build option.

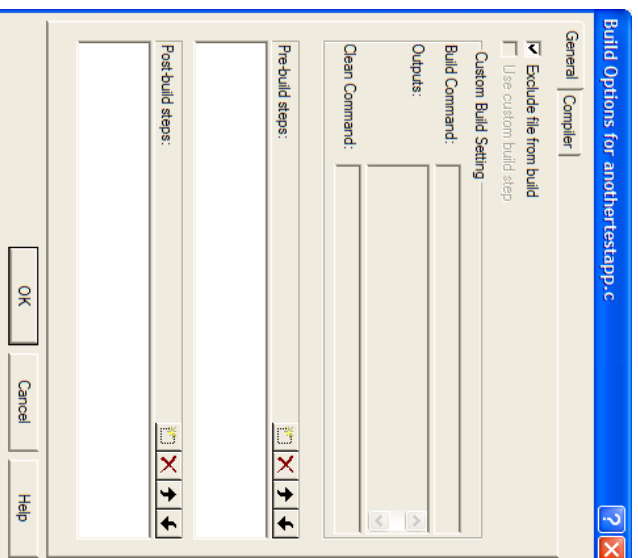
9. In the Project View window, right-click on anotherestapp.c and choose File Specific Options.
10. Select the General tab and check the Exclude File from Build option.

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007

Getting Started With the Code Composer Studio Tutorial

Page 34 sur 548



11. Click OK.

12. Right-click on mainapplication.pjt and choose Build. The project should build without errors, and function identically.




**Pre and Post Build Customization**

In this exercise you will customize the build process of a project or file. For example, during the build process, you may want to open a text file to detail any modifications made or perhaps you want to print helpful text messages in the Build Window. We can accomplish this using the Pre/Post Build Options.

### Project Level


Note that Initial Build Steps are completed before the compilation phase and Final Build Steps are completed after the build cycle.

1. In the Project View window, right-click on mainapplication.pjt and choose Build Options.
2. Select the General tab.

3. In the Initial build steps field, click the  icon and type the following line of text in the field that appears:


```
echo Building the Main Application
```

4. After typing the text, click in the open white field to complete the line.

5. In the Initial build steps field, click the  icon and add the following text:

```
set PATH
```

**Note:** WIN98 users should use "PATH" instead of "set PATH".

6. In the Final build steps field, click the  icon and add the following text:

```
echo Done Building the Application
```

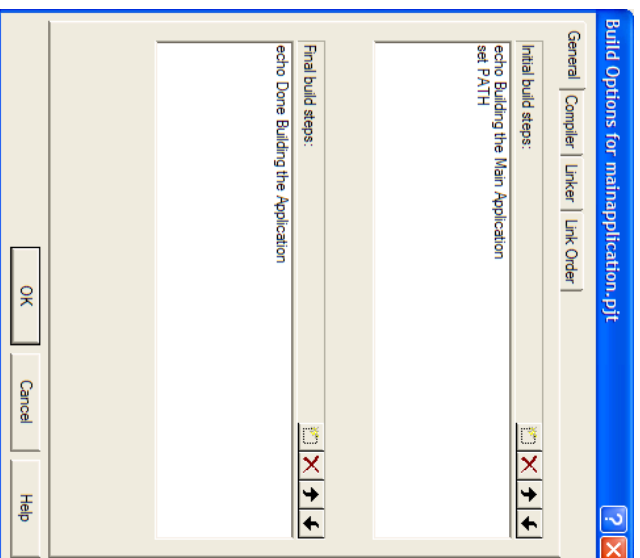
7. The dialog should resemble this:

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007

Getting Started With the Code Composer Studio Tutorial

Page 36 sur 548



8. Click OK to close the dialog and save your changes.

9. From the Project menu, choose Rebuild All. Notice the additional output displayed in the Build Window.



### File Level

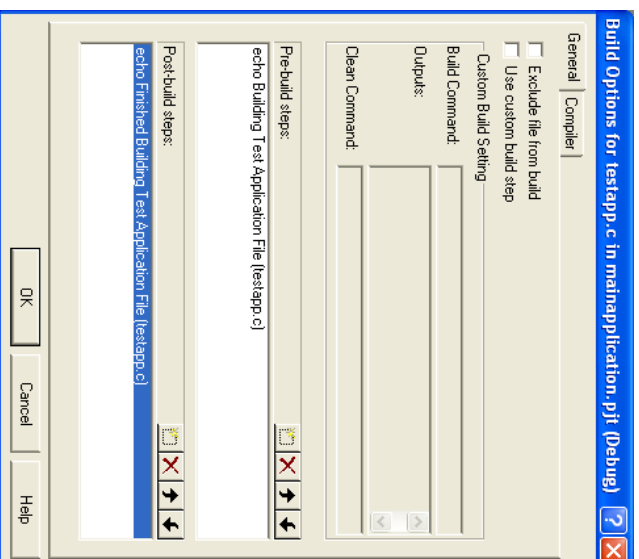
The same concept of Initial/Final Builds Steps can be applied to individual file compilation.

1. Right-click on testapp.c and select File Specific Options.

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007

2. Select the General tab.
3. In the Pre-build steps field, click the  icon and add the following text:  
echo Building Test Application File (testapp.c)
4. In the Post-build steps field, click the  icon and add the following text:  
echo Finished Building Test Application File (testapp.c)



file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007

Getting Started With the Code Composer Studio Tutorial

Page 38 sur 548

5. Click OK to close the dialog and save your changes.
6. From the Project menu, choose Rebuild All. Notice the additional output displayed in the Build Window.

### Custom Build Steps

Instead of using the default tool set for compiling files, you can now specify your own tools for object file generation using the Custom Build Setting option. In this section you invoke the Texas Instruments™ command-line compiler using the Custom Build Setting features.

First, you need to add additional functionality to the maxmlibrary to include averaging capabilities.

1. From the File menu, select **New**→Source File.
2. Select the following link to view the contents of averagevalue.c. Copy and paste the text into the source file you just created.:  
[averagevalue.c](#) (Click to view code)

3. From the File menu, choose **Save As**.
4. Browse to [C:\CCStudio\\_v3.10\myprojects\myapplication\](#).
5. Type **averagevalue.c** as the file name and click **Save**.
6. In the Project View window, select maxmlibrary.pjt and choose **Add Files to Project**.
7. Browse to [C:\CCStudio\\_v3.10\myprojects\myapplication\](#).
8. Select **averagevalue.c** and click **Open**.
9. In the Project View window, right-click on averagevalue.c and select **File Specific Options**.
10. Select the General tab and enable the checkbox next to the Use custom build step option.
11. In the Build Command field, enter the following text; be sure to include the quotation marks around the word debug:

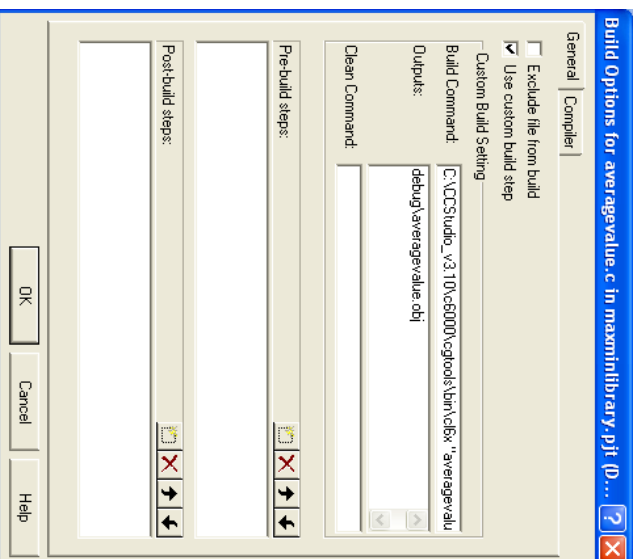
- `C:\CCStudio_v3.10\c6000\cgttools\bin\cl6x "averagevalue.c" -fr "debug"`
- Where `C:\CCStudio_v3.10\c6000\cgttools\bin\` is the path to the compiler
- `cl6x` is the name of the compiler

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007

- "averagevalue.c" is the name of the file
  - -fr "debug" is the option you want to use
12. In the Outputs text field, enter:
    - debug\averagevalue.obj

This is the location where the object file is placed after generation. This also ensures that the object file will be linked to the entire project.



13. Click OK to close the dialog and save your changes.

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007

Getting Started With the Code Composer Studio Tutorial

Page 40 sur 548

14. From the Project menu, choose Rebuild All.



### Viewing Multiple Projects

CCS - L2

#### Setting Active Projects

When working with multiple projects concurrently, it is often convenient to set a project as the active project. This ensures that all project commands (such as Build) are executed on the active project instead of all files in the workspace.

1. Right-click on mainapplication.
2. If it is not already set, select Set as Active Project. You can also do the same for maxminlibrary, to make it the active project. Notice that only one project can be active at a time. You can also set a project to active by selecting it from the drop-down menu above the Project View window.
3. Click on mainapplication.pjt in the Project View window. From the Project menu, choose Save.
4. From the Project menu, choose Close.
5. By default, when you have two projects open and close one, the remaining project is now active.
6. Click on maxminlibrary.pjt in the Project View window. From the Project menu, choose Save.
7. From the Project menu, choose Close. Close all open windows in the workspace as well.



### Using Version Control

CCS - L2

Before using the version control integration tools, create a version control project using your version control software. Consult your version control software documentation for further information on how to do this.

1. After creating a version control project with your software, add all the files from mainapplication.pjt and maxminlibrary.pjt, which you created earlier. If you followed the directions closely in the earlier exercises, both of the projects should be located in [C:\CCSStudio\\_v3.10\myprojects\myapplication](C:\CCSStudio_v3.10\myprojects\myapplication). Make sure you include all of the files (\*.pjt \*.c, \*.lib, \*.cmd).

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

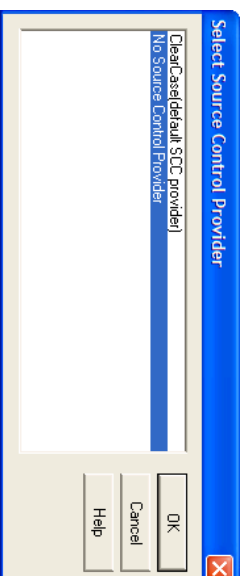
26/09/2007

**Note:** Library files are located in C:\CCStudio\_v3.10\66000\cgt001s\lib\.

2. Save any changes and Exit the version control application.

### Selecting a Provider

1. If you have not already done so, launch the Code Composer Studio™ IDE.
2. From the Project menu, choose Source Control→Select Provider. A dialog box similar to this will appear.



3. Select the desired Source Control Provider.

**Note:** The IDE automatically detects and lists any Source Control Providers in the dialog above.

4. Click the OK button to apply the setting.
5. Open the project file and source files that you have versioned in the previous section.

### Adding Files to Source Control

1. Right-click on a particular file and choose Add to Source Control. Depending on your version control software, you may be prompted with a version control software dialog to select a Project File. Select the project that you added to the control versioning software in the previous section. You will have to do this only once per project.
2. Depending on your version control software, you may be prompted with a dialog confirming your decision to add the file. Using that dialog, indicate that you would like to do this and press OK.
3. Repeat this procedure for all files within the project directory and all sub-directories (\*.pjt \*.c, \*.lib, \*.cmd).

### Accessing Provider Features

Once a file has been versioned correctly, you will have access to all of the features by choosing Project→Source Control. These features include:

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007

### Getting Started With the Code Composer Studio Tutorial

Page 42 sur 548

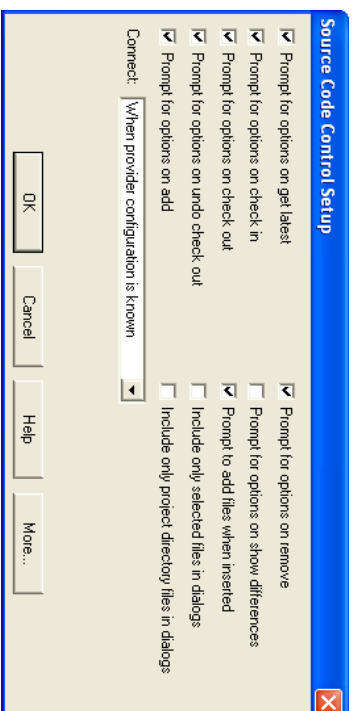
- Get Latest Version of source files
- Check Out source files
- Check In previously checked out source files
- Undo the last Check Out command
- Add file to Source Control
- Remove file from Source Control
- Show Differences between current file and checked-in file
- Show source file History
- Show Properties
- Refresh source file Status
- Select Source Control Provider (e.g. SourceSafe, ClearCase)
- Configure source control Options
- Launch source control Client Tool

### Additional Features

1. You can manually launch your source control software by choosing Project→Source Control→Launch Client Tool.
2. You can modify how to interact with the code version software by selecting Project→Source Control→Options. Most of the settings involve prompt messages when performing a particular action (e.g. Prompt for Options on Add.)

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007



## Multiple Project Configurations

CCS - L2

The Project Configuration feature allows you to set build settings at the project level. It also allows you to customize your build settings. Any build setting specified at this level applies to every file in the project.

1. From the Project menu, choose Open and browse to C:\CCStudio\_v3.10\myprojects\myapplication\mainapplication\.
2. Select mainapplication.pjt and click Open.

### Default Options: Debug and Release Settings

The Debug and Release settings allow you to have two custom configurations for compiler & linker settings. For example, you may prefer to build a slower application which has easy debugging capabilities in the debug setting, while the release setting might contain no debug information and thus be highly optimized.

**Note:** By default, when you create a project, a Debug and Release setting is created.

### Switching Between Options

1. From the Project menu, choose Configurations.
2. Select Debug (or Release) from the tree-like structure.

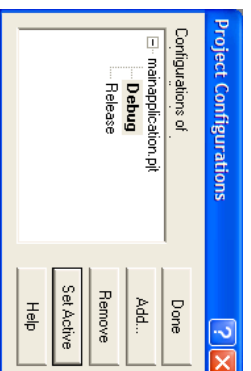
file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007

Getting Started With the Code Composer Studio Tutorial

Page 44 sur 548

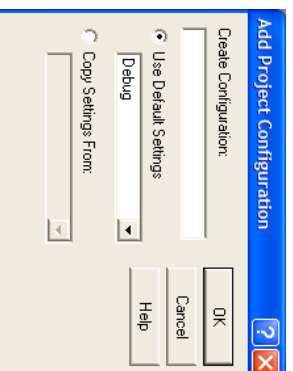
3. Click on the Set Active button. The selected configuration setting will appear boldfaced.



4. Click **Done** to close the dialog and save your changes.

### Adding New Project Configurations

1. From the Project menu, choose Configurations.
2. Click on the Add button. A dialog like the one below will appear.



3. Type a name for your new configuration in the Create Configuration text field.
4. Enable the radio button next to Use Default Settings.
5. Select Debug from the combo-box.

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007

- Click OK to save your changes and click Done to close the dialog, adding the new configuration to your project's configuration list. Your new configuration will be added to the configuration list for your project. For example, the following graphic shows the volume project with two configurations defined, Debug and Release.



**Note:** Now, when you make a project configuration active and define build options, those options are only used when that project configuration is active.

### Removing Project Configurations

- From the Project menu, choose Configurations.
- Choose the new configuration.
- Click on the Remove button.
- Click Yes to confirm the removal.
- Click Done to close the dialog.



### Setting Link Order

CCS - L2

Link order allows you to specify what order to link the object files generated from your project source code.

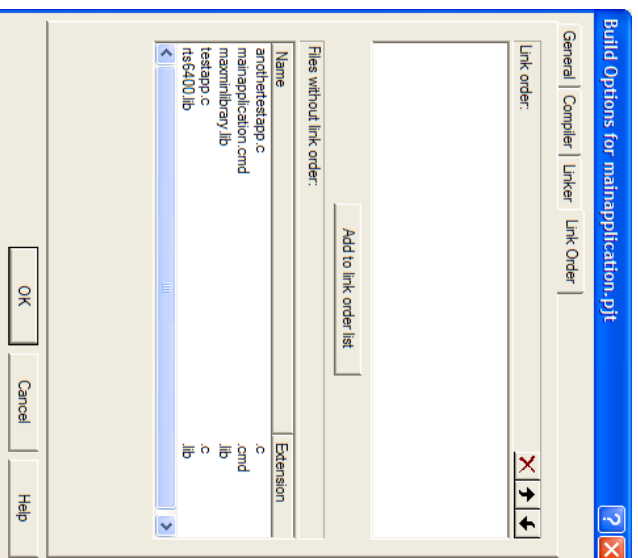
- In the Project View window, right-click on mainapplication.pjt and choose Build Options.
- Select the Link Order tab. A dialog box as shown below will appear.

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007

### Getting Started With the Code Composer Studio Tutorial

Page 46 sur 548



- Single-click on a file that you would like to specify a link-order for then press the Add to link order list button. Continue this for each file you would like to order.
- Single-click on a file and press the up and down arrows (top right, not on the keyboard) to change the order. Alternately, you can simply drag-and-drop the text labels to change the order. Press the Delete (X) icon to remove files from the list.
- Press OK to save your settings.



file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007

## Editing a Project File

CCS - L2

Although not recommended, you can manually edit your project file. This exercise shows you how to open, modify and change a project file.

1. In the Project View window, right-click on mainapplication.pjt and choose Open for Editing. This action will open the project file for viewing/editing.
2. Make your changes by editing in the file.

**Note:** For this exercise, do not make changes to this file.

3. From the File menu, choose Save.

If you made changes to your project file, the Save option will be available for you to select.

4. Once you save a project file, the program will ask if you would like to reload the project file. Choose Yes if you wish to reload.
5. From the File menu, choose Close.
6. Most of the text contains name-value pairs indicating project options and settings. Select the following link to view the contents of mainapplication.pjt: [mainapplication.pjt](#) (Click to view code)



## Creating an External Makefile

CCS - L2

You can create a Makefile using an external application like Microsoft Visual C++. You can then import that Makefile.

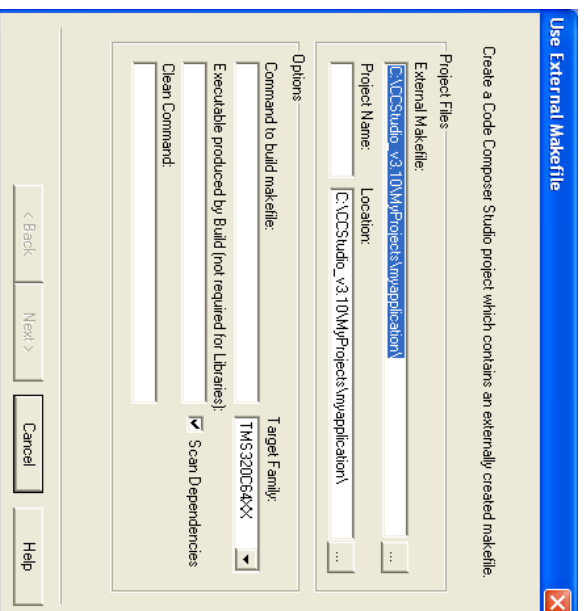
1. Locate or generate a Makefile file using another software development tool such as Visual C++. View the specific development tool documentation for details.
2. From the Project menu, select Use External Makefile. A dialog like the one below will appear.

file:///C:/Documents and Settings/Pierre-Armand/Local Settings/Temp/~hhBA53.htm

26/09/2007

Getting Started With the Code Composer Studio Tutorial

Page 48 sur 548

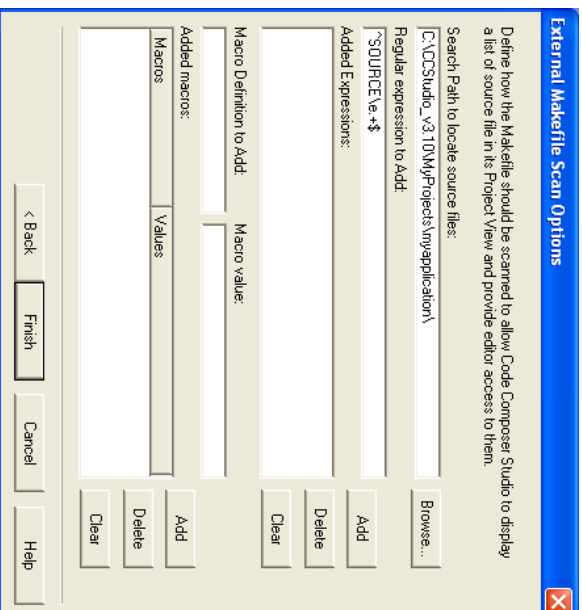


3. In the External Makefile text field, use the Browse button to locate the makefile created previously.
4. Change the Target Family option to point to your target board.
5. Press Next. The following dialog will appear.

file:///C:/Documents and Settings/Pierre-Armand/Local Settings/Temp/~hhBA53.htm

26/09/2007





Don't press Finish just yet!. In the next section, you will learn how to use the regular expression tool.

### Using Regular Expressions

A regular expression defines a specific text pattern that can be used to locate and retrieve specific sections of text. For example, `nat.*` is a regular expression which searches for all occurrences of "nat" followed by one or more characters (e.g. nathan, nate, natalie, etc.). There are many more regular expression characters, however, these are the most commonly used ones.

### Regular Expressions By Example

Expression	Meaning
dsp	All occurrences of "dsp"

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007

### Getting Started With the Code Composer Studio Tutorial

Page 50 sur 548

^dsp	All occurrences of "dsp" which occur at the beginning of a line.
<dsp\$	All occurrences of "dsp" which occur at the beginning of a line and end at the end of the line (e.g. the only thing on the line)
.	Matches any single character, 1 or more times
*	Matches any character, 0 or more times
+	Matches any character, 1 or more times

### Adding Regular Expressions to Makefile Import

In this exercise, you will learn how to specify the source files you wish to import within the makefile importing utility. The makefile utility will scan the makefile you are importing, looking for the source files you specify. In this exercise, you will configure the makefile utility to scan for source files and include files. Specification is done using regular expressions as in the previous exercise.

1. Add the default regular expression `"^SOURCE\w+$"` (without the quotes) to the Added Expression section by clicking the Add button. In this instance, `"^SOURCE\w+$"` indicates that CCS will search the makefile for all lines containing source file information (e.g. `SOURCE=<path>`). The `"^"` and `"$"` characters indicate that these lines must have no additional or extraneous text.
2. In the Regular Expression to Add, type `"^INCLUDE\w+$"` (without the quotes). Press Add. In this instance, `"^INCLUDE\w+$"` indicates that CCS will search the makefile for all lines containing header file information (e.g. `INCLUDE=<path The "<w>" and "<w>" characters indicate that these lines must have no additional or extraneous text.`

Don't press Finish yet! In the next exercise you will learn how to use the macro tool.

### Using Macros

Macros allow you to apply advanced search-and-replace techniques to a file. For example, below is a macro definition located in a makefile:

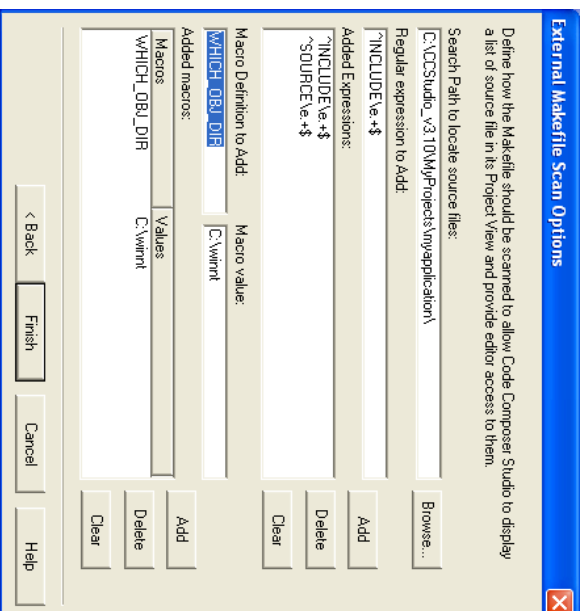
```
WHICH_OBJ_DIR \
    ^$(INTDIR)\application.obj* \
    ^$(INTDIR)\function1.obj* \
    ^$(INTDIR)\function2.obj* \
    ^C:\winnt\application.obj* \
    ^C:\winnt\function1.obj* \
    ^C:\winnt\function2.obj* \
```

If we define `WHICH_OBJ_DIR` to equal `"C:\winnt"` (without quotes) within the makefile import tool, the generator will add the following text into the generated project file (.pj1):

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007

1. In the Macro Definition to Add field, enter: WHICH\_OBJ\_DIR
2. In the Macro value field, enter: C:\wint
3. Click the Add button to add the macro. Your window should look like the one below.



**Note:** You must examine your own makefile to identify any macros defined within it. Each software development tool defines its own set of macros.

4. Click Finish to save.



### Generating an External Makefile

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007

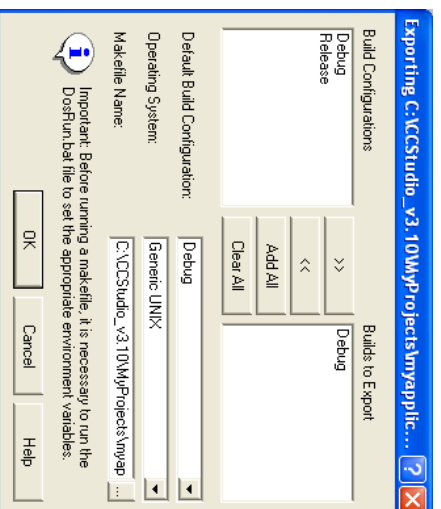
Getting Started With the Code Composer Studio Tutorial

Page 52 sur 548

CCS - 12

In this exercise you will learn how to export a project file (.pj1) file to a makefile.

1. Select mainapplication.pjt and set it as the active project.
2. From the Project menu, choose Export to Makefile. A dialog such as the one below will appear.



3. In the Build Configurations field, select the project configuration you would like to generate a makefile from by selecting it with the mouse. For this exercise, select Debug.
  4. Click the >> arrow to add the configuration to the Builds to Export field.
  5. In the Operating System field, select Generic Unix.
  6. In the Makefile Name field, click the browse button and browse to the location you want to create your external Makefile in. By default, the name and location is the same as the active project.
  7. Press OK. A makefile will be generated and placed into the specified folder.
- This concludes the Project Management lesson.

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007



## Editing Techniques

---

CCS - L3

In this lesson, you will learn how to use Advanced Editing features in the context of modifying a pre-existing DSP project.

Learning Objectives:

- Use bookmarks to locate specific sections of code
- Use an external editor to change project code for Code Composer Studio™ IDE
- Customize the Code Composer Studio editor

Example used in this lesson: `datedisplay (mainapplication.pjt` and `maxminlibrary.pjt)`

Application Objective:

This lesson uses a DSP project to demonstrate advanced editing features, including: code bookmarking, external editors, and custom extensible keyword highlighting.

This lesson consists of the following steps. To go directly to a step, click on the link below:

- [Getting Started](#)
- [Using Bookmarks](#)
- [Column Editing](#)
- [Using an External Editor](#)
- [Extensible Keyword Highlighting](#)
- [Using CodeSense](#)
- [Using the Selection Margin](#)
- [File Editing Tips](#)

[file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm](#)

26/09/2007

## Getting Started With the Code Composer Studio Tutorial

Page 54 sur 548



The forward arrow will take you to the next page in this lesson.

## Getting Started

---

CCS - L3

Over the course of the lesson, you will use the following types of files:

- `.lib` This library provides runtime support for the target DSP chip
- `.c` This file contains source code that provides the main functionality of this project
- `.pjt` This file contains all of your project build and configuration options
- `.kwd` This file implements extensible keyword highlighting

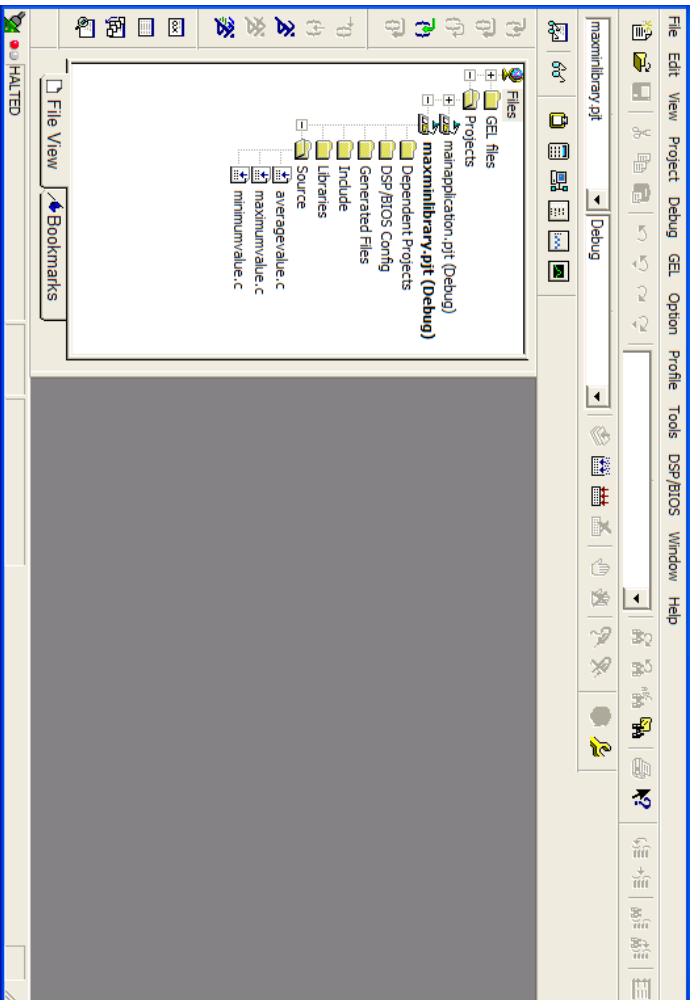
First, open the two projects in the `datedisplay` example.

1. From the Project menu, choose Open.
2. Browse to `C:\CCStudio_v3.10\tutorial\target\datedisplay\` select `mainapplication.pjt` and click Open.
3. Browse to `C:\CCStudio_v3.10\tutorial\target\datedisplay\` select `maxminlibrary.pjt` and click Open.

Notice that you now have two projects open in the project view window. You can select which project is active by selecting it from the project selection drop down menu above the project view window. When you build a project, a `.out` file is built. By default, the program creates a project configuration containing a debug directory for you to build your `.out` file into. You can change the project configuration (and the location the `.out` file is built into) by selecting debug or release from the drop down dialog above the project view window.

[file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm](#)

26/09/2007



4. From the project selection drop down menu, choose maximilibrary.pjt:



5. From the Project menu, choose Rebuild All.

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007

Getting Started With the Code Composer Studio Tutorial

Page 56 sur 548

6. Repeat step 4 and 5 for 'manapplication.pjt'.

You can also build your application by right-clicking on it in the project view window and choosing Build. When using this method, the project is re-linked, and a new .out file is created.



### Using Bookmarks

CCS - L3


In this exercise, you will learn how to use file bookmarks. File bookmarks allow you to bookmark specific lines in your project files for quick and easy locating.

1. From the Edit menu, choose Bookmarks. A dialog box will open.
2. Click the Add button. A dialog box such as the one below will open.



3. Click the Browse button. Browse to and select maximumvalue.c from the project directory. Click Open.
4. In the Line field, enter 10 and click OK. Optionally, you can add a description of your bookmark in the Description field.

**Note:** The description you enter will be displayed as the caption for the bookmark. Otherwise the file and line number are displayed.

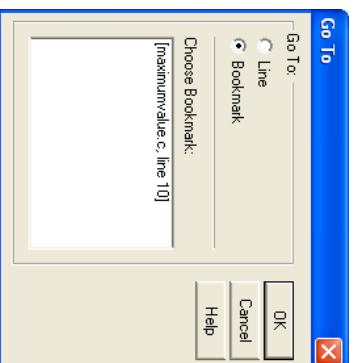
5. In the Bookmarks dialog, enable the Show Bookmarks checkbox. This option places a flag icon  within the document window to indicate where bookmarks have been placed. You can see this when you open a bookmarked file.
6. Experiment with creating various bookmarks within the code.

You can add, edit, remove, or go to bookmarks by selecting the bookmark in the Bookmark dialog and choosing Add, Remove, Edit, or Go To. You can also go to bookmarks by choosing Go To from the Edit menu. For the Go To menu item to be active, you must first open a bookmarked source file.

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007

7. When you are finished experimenting, click Close.
8. In the maximilibrary project, open maximumvalue.c by double clicking on it.
9. From the Edit menu, choose Go To. A dialog box will open.
10. Enable the radio button next to Bookmark. A list of any bookmarks present in the active source file will appear in the Choose Bookmark field. The dialog will change as indicated below.



11. Select a bookmark from the Choose Bookmark field and click OK. The IDE will open the appropriate source file and place the cursor on the bookmarked line.

### Column Editing

CCS - L3

In this exercise you will learn how to use the Column Editing features. Column editing must be enabled to use this feature.

1. Right-click on mainapplication.pjt and choose Add Files to Project.
2. Browse to C:\CCSStudio\_v3.1.0\tutorial\Target\datetdisplay\.
3. Select usingcolumnediting.c and click Open.

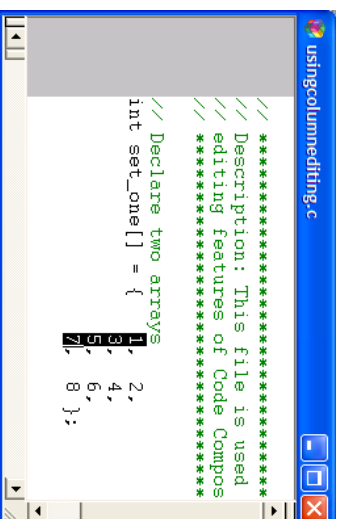
file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007

### Getting Started With the Code Composer Studio Tutorial

Page 58 sur 548

4. In the Project View window, open the Source folder, and double-click on usingcolumnediting.c to open it.
5. Using the mouse, place your cursor in front of the number 1 in usingcolumnediting.c.
6. Use the right mouse button to highlight the column of numbers as shown below.



7. Try copying the column once you select it by pressing Ctrl + C.
8. Now try pasting it by pressing Ctrl + V. Experiment with cutting, copying, pasting and deleting columns.

### Using an External Editor

CCS - L3

In this section you will learn how to integrate your program with an existing external editor application. The external editor interacts with the Code Composer Studio™ environment using command-line options and switches. These options and switches can be stored (and later reloaded) using editor configuration files (.ini files).

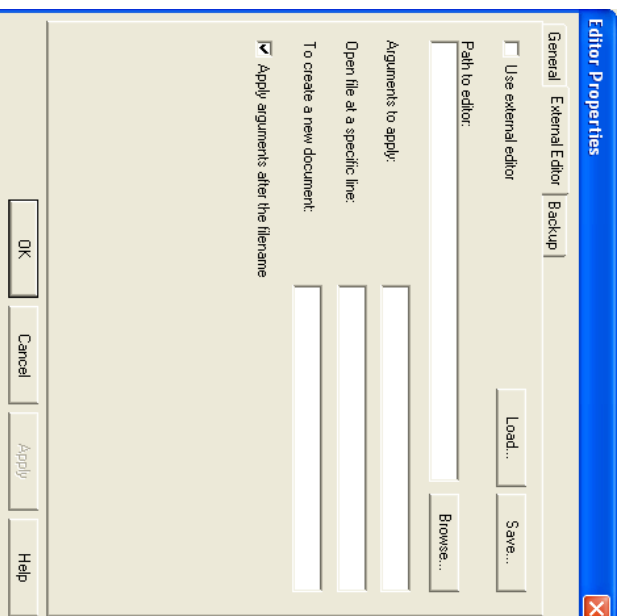
**Note:** In this example we will use an application called CodeWright™. (We also assume that you have installed the application properly on your computer). Other applications could also be used.

### Using Existing Editor Configurations

1. From the Option menu, choose Editor→Properties. Click on the External Editor tab. Your dialog will resemble the one below.

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007



2. Enable the check box next to Use external editor.
3. Click the Load button and browse to C:\CCStudio\_v3.10\cc\bin\editor\external editor\.
4. Locate the file named *CodeWright.ini* and click Open.
5. Click the Browse button and browse to the location of the executable for your external editor, and click Open.
6. Click OK.
7. From the File menu, choose New→Source File. This action should create a new file using the external editor specified in the previous steps.

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007

Getting Started With the Code Composer Studio Tutorial

Page 60 sur 548

### Creating New Editor Configurations

In this sub-section you will learn to create your own configuration file for an external editor. In this example, we create a configuration file for CodeWright™. Consult your editor documentation for details on how to do this.

1. Launch your favorite text editor. (In this example we will use Windows Notepad).
2. Create a file called mycodewright.ini and enter the following information:

```
[Settings]
Path=C:\CW2\CW2.exe
ArgsOpenAtSpecificLine=-g
ArgsCreateNewDocument=-x MenuCmd 'Document' 'New Document' '
ArgsAfterFilename=1
```

- Path: Indicates the location of the application to launch
  - ArgsOpenAtSpecificLine: Indicates the command-line arguments to use when the IDE needs to open a file at a specific line number
  - ArgsCreateNewDocument: Indicates the command-line arguments to use when the IDE needs to open a new file
  - ArgsAfterFilename: Indicates the number of arguments that follow the specified file name
3. Save this file and load it as the external editor customization file, as before. This external editor customization file will load CodeWright in the same manner as before except that the "splash screen" for CodeWright will be displayed.
  4. To continue the lesson, we must stop using an external editor, so from the Option menu, choose Editor→Properties. Click on the External Editor tab and un-check the Use external editor box.



### Extensible Keyword Highlighting

CCS - L3

In this exercise you will learn how to create a custom file to implement extensible keyword highlighting.

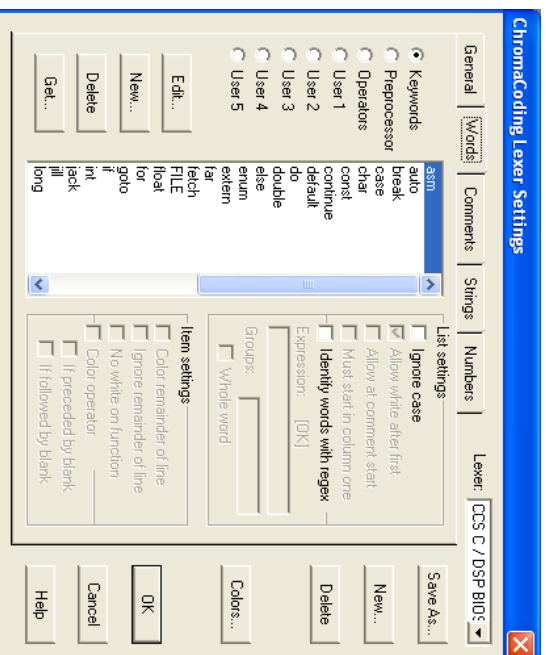
file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007

1. Launch your favorite text editor (In this example we will use Windows Notepad).
2. Create a file called mykeywords.txt and type the following information :

```
File Edit Format View Help
1ack
1111
Fetch
water
```

3. Save the file to C:\CCStudio\_v3.10\cc\bin\editor\ folder.
4. From the Option menu, choose Editor→ChromaCoding Lexers, then select the Words tab.



5. Click the Keywords radio button, if it is not already selected.

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007

Getting Started With the Code Composer Studio Tutorial

Page 62 sur 548

6. Click the Get button. Browse to and select mykeywords.txt and click Open.
7. Click OK.
8. Open a .c extension file.
9. Type one of the keywords you specified in your custom file. The editor window should highlight the word.

**Note:** You can also add keywords to the list by clicking on the New button and typing the words separated by spaces.



### Using CodeSense

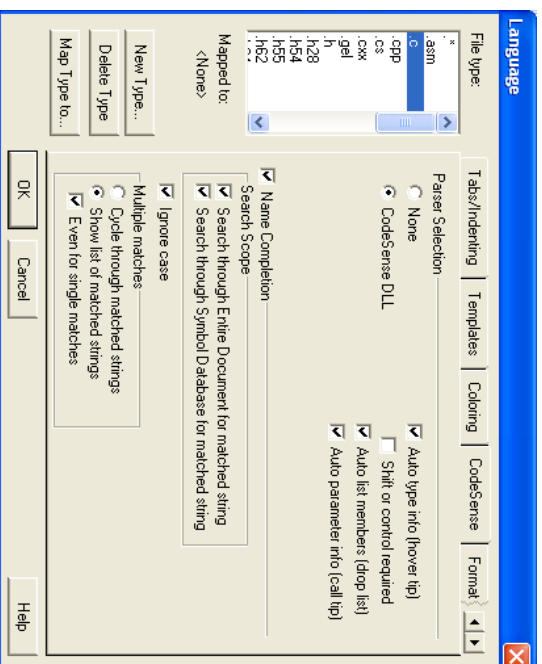
CCS - L3

In this exercise you will learn how to set up and use CodeSense Settings.

1. Right-click on mainapplication.pjt and choose Add Files to Project.
2. Browse to C:\CCStudio\_v3.10\tutorial\Karget \katedisplay\.
3. Select usingcodesense.c and click Open.
4. In the Project view, select usingcodesense.c and click Open.
5. From the Option menu, choose Editor→Language, then choose the CodeSense tab. You may have to use the arrow buttons on the side to find the appropriate tab. You should see a dialog like the one below.

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007



6. Ensure that all checkboxes are check-marked as shown.
7. Click OK.
8. In the project view window, double-click on usingcodesense.c to open it.

#### Suggesting Words (Active)

1. Type the letter C at the bottom of the file, then press the Ctrl key and the space bar together.
2. A drop-down list will appear, suggesting various possible word completions. Your list members may vary.

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007

Getting Started With the Code Composer Studio Tutorial

Page 64 sur 548



3. Select one of the choices with the mouse to complete the word.

**Note:** You may ignore the suggestion by simply continuing to type.

#### Listing "Member" Information

1. Type the following text at the end of the file:
  - futureDate.

**Note:** Until you type the period, no drop box appears.

2. A drop-down box will appear, listing all the member variables of the C-structure.
3. Select any member variable.

#### Displaying Parameter Information

Type the following text at the end of the file:

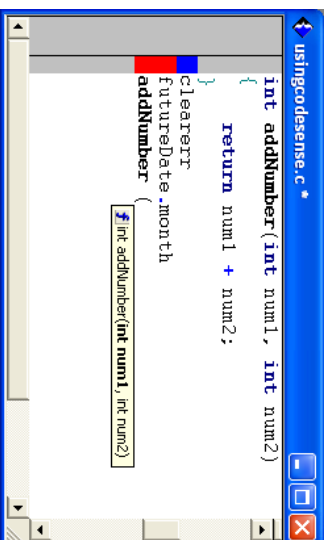
```
addNumber (
```

The open parenthesis starts a pop-up box to suggest the parameter information for the addNumber(int, int) function. This ensures the correct order of function parameters, as shown in the following example.

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007





**Note:** If a particular function is overloaded, you can click on the pop-up box to view each overloaded function's parameter information.



### Using The Selection Margin

CCS - L3

In this exercise you will learn how to customize and use the selection margin.

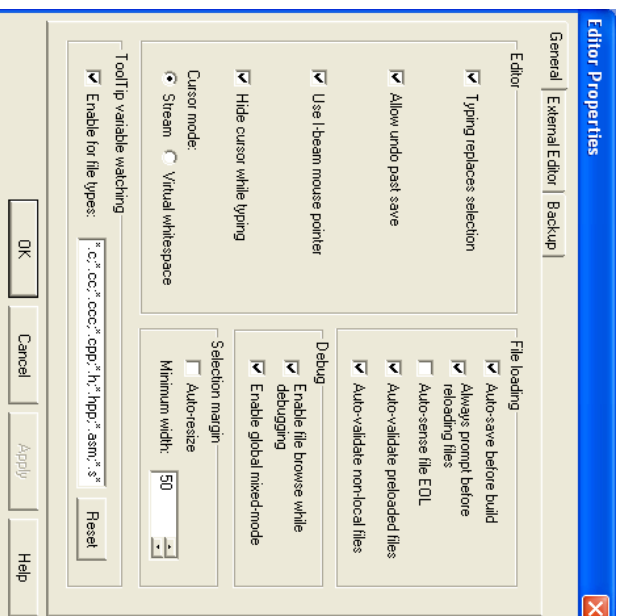
1. Open a .c extension file by double-clicking on it in the Project View window.
2. From the Option menu, choose Editor→Properties, then select the General tab.
3. In the Selection margin section, you can select Auto-resize to automatically re-size the selection margin to fit all the information, or specify a minimum width.
4. Set the Minimum width to 50.

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

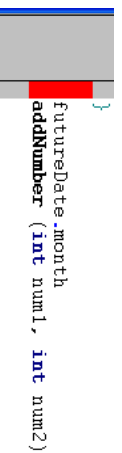
26/09/2007

### Getting Started With the Code Composer Studio Tutorial

Page 66 sur 548



5. Press **OK**. Your left margin should be resized, as shown in the following example.



file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007



---

## File Editing Tips

CCS - L3

### Assembler Command Help

In this section, you will learn how to access help options on assembler commands. This is particularly useful when debugging an application.

1. From the File menu, choose Load Program. Select any .out file and click OK.
2. A window will open containing the program assembly instructions. Maximize this window.
3. Single-click on any assembly instruction (not the entire line – just the instruction!). Press F1. A help application will open, containing further documentation on your selected assembler instruction.

### Modifying Existing Configuration Files

You can customize the command-line options for an existing external editor by opening an existing .ini configuration file and modifying the text contained in the *Arguments field*.

1. From the File menu, choose Open, and browse to [C:\CCStudio\\_v3.10\cc\bin\Editor\External Editor\](file:///C:/CCStudio_v3.10/cc/bin/Editor/External Editor/).
2. Select the .ini file for your external editor.
3. Make your desired changes to the file and click Save.
4. From the Option menu, choose Editor→Properties, the select the External Editor tab.
5. Click the Load button.
6. Select the .ini file for your external editor and click OK.

This concludes the Editing Techniques lesson.



---

## Using Debug Tools

CCS - L4

<file:///C:/Documents and Settings/Pierre-Armand/Local Settings/Temp/~hhBA53.htm>

26/09/2007

---

## Getting Started With the Code Composer Studio Tutorial

Page 68 sur 548

In this tutorial you will learn how to use the debugging features within the context of a DSP project that you will create.

### Learning Objectives:

- Use Probe Points to inject and extract data from your running application
- List the four ways that Probe Points and Breakpoints differ
- Identify the functions and uses of the Symbol Browser
- Create tabs and expressions in the Watch Window to examine the program
- Create, manage and build projects quickly using a command line interface in the Command Window

Examples used in this lesson : sinewave

### Application Objective:

While creating a DSP application to apply a gain factor to a sine wave, you will learn how to use Probe Points to accept input from a file, and how to examine your program using the Symbol Browser. Finally, you will learn how to simulate debugging an actual DSP chip using board interrupts.

This lesson consists of the following steps. To go directly to a step, click on the link below:

### [Reviewing the Source Code](#)

[Using Probe Points](#)

[Using the Symbol Browser](#)

[Using the Watch Window](#)

[Using the Command Window](#)



The forward arrow will take you to the next page in this lesson.

---

## Reviewing the Source Code

CCS - L4

<file:///C:/Documents and Settings/Pierre-Armand/Local Settings/Temp/~hhBA53.htm>

26/09/2007

1. From the Project menu, choose Open. Browse to C:\CCStudio\_v3.10\tutorial\target\sinewave\.
2. Select sinewave.pjt and click Open.
3. In the Project View window, click on the + sign next to sinewave.pjt.
4. In the Project View window, click on the + sign next to Include, Libraries, and Source.

Here is a list of all files and their purpose:

- rts.lib This library provides runtime support for the target DSP chip
  - sine.c This file contains source code that provides the main functionality of this project
  - sine.h This file declares the buffer C-structure as well as define any required constants
  - SineWave.pjt This file contains all of your project build and configuration options
  - SineWave.cmd This file maps sections to memory
5. In the Project View window, double-click on sine.c to open it.

Note the following salient features of the source code:

- The function first prints a message "sinewave example started" (without the quotes).
- The first function entered is the dataIO() function. This function is currently empty. It will be "connected" later to input/output files using Probe Points.
- The second function entered is the processing() function. This function takes the input values and applies a preset gain factor to the value.
- This process is repeated until the user terminates the program.

Take a minute to examine the code. Examine the parameters the functions take. Examine the flow of program control. Can you provide a one line description of what the program does?



### Using Probe Points

CCS - 14

In this exercise you will learn how to use Probe Points. Probe Points allow you to inject and extract data from your running application. This injection of data typically takes the form of an input file. The extraction can take the form of a graph (within the program) or an output file. Probe Points can also be used to refresh windows with updated data. There are both software

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007

## Getting Started With the Code Composer Studio Tutorial

Page 70 sur 548

and hardware probe points, but this tutorial only uses software probe points. Hardware probe points are implemented by the hardware internally, and can be set in any memory type. They are typically used when software probe points will not work, such as in memory that cannot be written to, like ROM memory. Software probe points are implemented as an opcode replacement, and there is no limit to the number of software probe points that can be set.

**Note:** If you are using a DSK or EVM, you must establish a connection via Debug=Connect before loading the program.

### Setting up an I/O Probe Point

1. From the Project menu, choose Rebuild All.
2. From the File menu, choose Load Program. Browse to C:\CCStudio\_v3.10\tutorial\target\sinewave\Debug\.
3. Select sinewave.out and click Open.
4. In the Project View window, double-click on sine.c. Place your cursor on dataIO() inside the main() function (line 30). The actual line number may vary.

```

void main()
{
    puts("SineWave example started.\n");
    while(TRUE) // loop forever
    {
        /* Read input data using a probe-point connected to a host file
        Write output data to a graph connected through a probe-point
        dataIO();
        /* Apply the gain to the input to obtain the output */
        processing();
    }
}

```

You will now inject data from a text file into your program.

5. Right-click on the dataIO() function and choose Toggle Software Probe Point. This action puts a blue diamond in the margin of your C source file to indicate a Probe Point.
6. From the File menu, choose File I/O. The File I/O dialog box will appear, allowing you to select input and output files.
7. Select the File Input tab and click Add File.

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007

8. Browse to `C:\CCStudio_v3.10\tutorial\target\sinewave\`.
9. Select `sin.dat` and click `Open`. This file contains data points encoded in hexadecimal form for a sine curve. Notice that in the File Input tab, you can select data files of integer, hexadecimal, long and float types.
10. A tape-player control like the one below will open. You can use this window to start, stop, rewind and fast-forward within your data file while running your program.



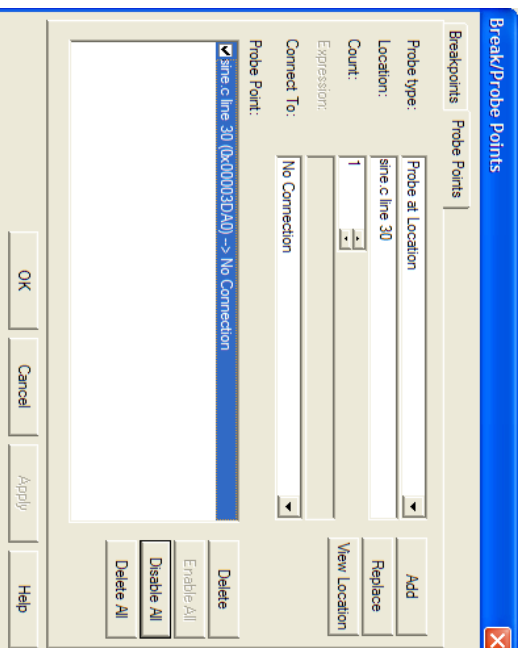
11. In the File I/O dialog, change the Address to `currentBuffer.input` and change the Length to 100. This means that the Probe Point will "inject" 100 data points into `currentBuffer.input` each time it hits the Probe Point.
12. Ensure the Wrap Around option is check-marked. This means that the 100 data points will continuously be cycled through.
13. Click on the Add Probe Point button. A dialog like the following will appear, although the actual line number may vary.

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007

Getting Started With the Code Composer Studio Tutorial

Page 72 sur 548



14. In the Probe Point field, select the Probe Point you created previously (`sine.c line 30`). The actual line number may vary.
15. In the Connect To drop-down list, select the `sin.dat` file.
16. Click `Replace`. Click `OK`. Click `OK` again. You have now successfully created an input Probe Point.

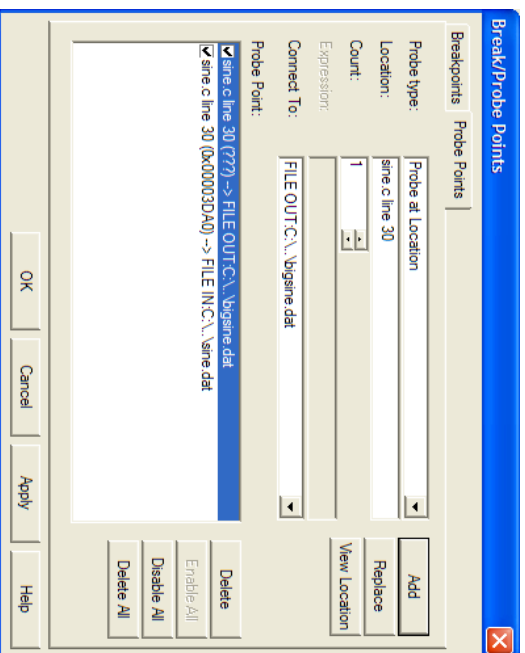
Now you'll create a Probe Point at the same point. This time, however, the Probe Point will be an output file.

1. From the File menu, choose File I/O.
2. Select the File Output tab, and click `Add File`.
3. Browse to `C:\CCStudio_v3.10\tutorial\target\sinewave\`.
4. Select `big.sine.dat` and click `Open`.

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007

5. Change the Address to currentBuffer.output and the Length to 100.
6. Click Add Probe Point.
7. In the Probe Point field, select the Probe Point you created previously (sine.c line 30). The actual line number may vary.
8. In the Connect To drop-down list, select bigsine.dat and click Add. The Breaky/Probe Points window should look like this, although the actual line number may vary:



9. Click OK. Click OK again. You have now successfully created an output Probe Point.
10. From the Debug menu, choose Run. The "cassette-player" controls should start "playing". At this time, the input is being injected and the output is being extracted.

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007

Getting Started With the Code Composer Studio Tutorial

Page 74 sur 548



**Note:** While running the program, the output file bigsine.dat will increase in size. Once you understand how the program functions, please stop program execution. Also, if you are using hardware DSK, the system will stop reaching each probe point, and so must be restarted by selecting Run from the Debug menu.

11. From the Debug menu, choose Halt.
12. Now, open bigsine.dat to verify it contains the original sinewave with the gain factor applied.

**Note:** You can also display data from bigsine.dat visually, using a graph. For more information, see the Data Visualization lesson.

### Probe Points vs. Breakpoints

Probe Points and breakpoints both halt the target to perform actions. However, Probe Points differ from breakpoints in the following ways:

- Probe Points halt the target momentarily, perform a single action, then resume target execution
- Breakpoints halt execution until execution is manually resumed
- Breakpoints cause all open windows to be updated
- Probe Points permit automatic file input/output, while breakpoints do not



### Using the Symbol Browser

In this exercise, you will use the symbol browser to examine the SinWave project.

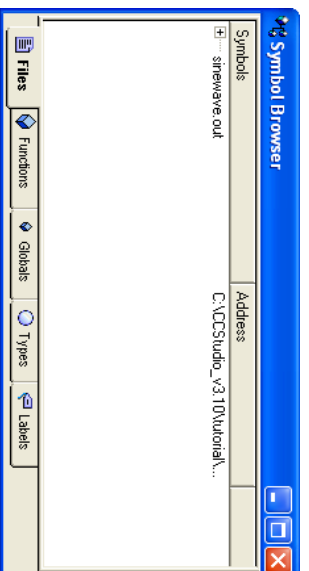
1. From the Project menu, choose Rebuild All.
2. From the File menu, choose Load Program.
3. Browse to C:\CCStudio\_v3.10\tutorial\target\sinewave\Debug\.
4. Select sinewave.out and click Open.

CCS - L4

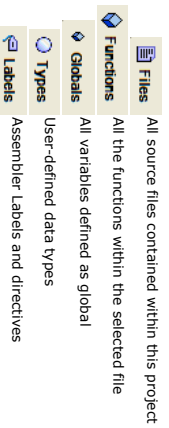
file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007

- From the View menu, choose Symbol Browser.



- The Symbol Browser allows you to examine the project by selecting the following tabs :



- Experiment with the Symbol Browser by selecting one of the tabs and expanding lists by clicking on any + sign. Click on the Functions tab and expand the list.
- Click on a function in the list. This will open the sinewave.c file to the location of the function in the file.
- When you are finished, right-click in the Symbol Browser window and choose Close.



### Using the Watch Window

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007

Getting Started With the Code Composer Studio Tutorial

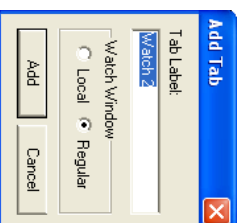
Page 76 sur 548

CCS - 14

The Watch Window allows you to create custom Watch Tabs. Within these tabs you can define groups of variables. For example, you can create a Watch Tab for all pointers, and another Tab for all member variables.

#### **Creating a Tab**

- From the View menu, choose Watch Window.
- Right-click on the Watch Window text pane and select Add Tab. An Add Tab dialog will appear.



- In the Tab Label field, enter *debug 1* as the name for the tab.
- Enable the radio button next to Regular.
- Click Add. The Watch Window should show the new tab and look like this:



file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm


26/09/2007

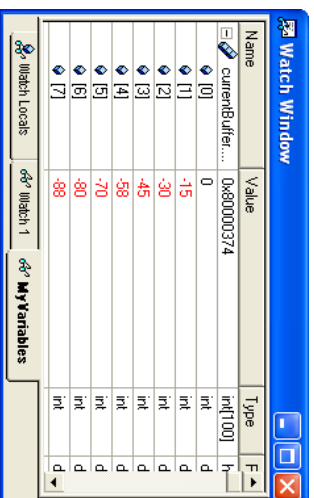
### Deleting a Tab

1. Select the tab you just created.
2. Right-click in the Watch Window and select Delete Tab. A confirmation dialog will appear.
3. Click OK to confirm.

### Adding an Expression

Using the directions above, add a tab named MyVariables.

1. Click on the expression icon  in the Name column and type currentBuffer:input as the name of the expression to watch.
2. Click on the white space in the watch window to save the change. The value should immediately appear.
3. From the Debug menu, choose Run.
4. Expand the input array by clicking on the + sign next to currentBuffer:input.
5. Observe the changing values in the Watch Window.



6. From the Debug menu, choose Halt.

### Deleting an Expression

1. Select the expression you wish to remove from the Watch Window by clicking on it with the mouse.

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007


Getting Started With the Code Composer Studio Tutorial

Page 78 sur 548

2. Press the Delete key on the keyboard. The expression and all sub-expressions are removed from the Watch Window.

### Changing View Format

Formatting symbols are used to change the display of the variables in the Watch Window (e.g. d=Decimal, f=Decimal Floating Point, etc.)

1. Select the MyVariables tab.
2. Click on the expression icon  in the Name column and type currentBuffer:input as the name of the expression to watch.
3. Click on the white space in the watch window to save the change. The value should immediately appear.
4. Click on the Radix heading in the Watch Window. A drop down list will appear.
5. Select a new format from the list. You can also select a new format for a single value by clicking in one of the Radix fields and selecting a new format from the list.

**Note:** If the Radix column is not visible, expand the Watch Window until it is visible.



### Using the Command Window

CCS - L4

The Command Window allows you to execute a limited set of MS-DOS commands along with Texas Instruments (TI) High Level Language (HLL) commands. These HLL commands match those found in previous versions of the TI debugger. These commands allow you to create, manage and build CCS projects quickly using a command line interface. This lesson will briefly cover a few of the tasks you can perform with the command window:

### Opening an Existing Project

1. From the Tools menu, choose Command Window.
2. In the Command field, use MS-DOS commands to change the directory to the location of the project you want to open.
3. In the Command field, type project open projectname.pjt. This should open the project you specify in the command.

### Building a Project

In the Command field type project buildall. This should build the entire project.

**Note:** Before going on to the next lesson, remove all Probe Points.

This concludes the Using Debug Tools lesson.

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007



## Data Visualization

---

CCS - L5

Data visualization is often the most effective method of debugging a DSP application. The variety of graphical data formats allows us to gain great insight into the operation of algorithms. In this tutorial, we will examine this important area of the DSP code debugging process, as facilitated by the capabilities of Code Composer Studio™ IDE.

[More About Data Visualization](#) (Click to view additional information)

Learning Objectives:

- Use Probe Points and File I/O to facilitate data visualization
- Identify how various DSP visualization schemes display the execution of code
- Create four common DSP visualization diagrams

Examples used in this lesson: modern

Application Objective:

You will set up graphical representations of data in the modern transmitter application. Following the setting up of the data file inputs (for modern data and signal noise), you will investigate how this input affects the execution of code. You will set up 4 different graphical displays, including an amplitude/time plot, an FFT waterfall, an eye diagram, and a constellation diagram.

This lesson consists of the following steps. To go directly to a step, click on the link below:

[Setting up for Data Visualization](#)

[Amplitude vs Time Diagram](#)

[Eye Diagram](#)

[Constellation Diagram](#)

[Fast Fourier Transform \(FFT\) Diagram](#)



file:///C:/Documents and Settings/Pierre-Armand/Local Settings/Temp/~hhBA53.htm

26/09/2007

## Getting Started With the Code Composer Studio Tutorial

Page 80 sur 548

The forward arrow will take you to the next page in this lesson.

### Setting up for Data Visualization

CCS - L5

This part of the tutorial will examine the capabilities of the program beyond the "traditional" scope of mainstream debuggers. Sample data will be used to test the execution of an algorithm and how such investigation lends insight into the operation of DSP code on target systems.

DSP applications are usually data-specific and therefore difficult to debug through exclusive use of the stepping/watching capability discussed in [Developing a Simple Program](#). One of the best methods of debugging code is to use sample data to observe and investigate the effect of the data on your code. Because of the application-specific nature of DSP code, input files containing sample data can provide a more functional representation of the algorithm by permitting the developer to observe how the written code changes known data.

For your convenience, a "modern" application has been set up as a Code Composer Studio Project. Before you start building a Data Visualization Diagram, you need to open the project and define a Probe Point and input data for the diagram. You will use and create various files during the lesson, as follows:

- .lib This library provides runtime support for the target DSP chip
- .c This file contains source code that provides the main functionality of this project
- .jit This file contains all of your project build and configuration options
- .dat This is a data file

1. From the Project menu, choose Close to close any projects that might already be open.
2. From the Project menu, choose Open.
3. Browse to [C:\CCStudio\\_v3.10\tutorial\target\modern\modern.jit](#) and click Open.
4. From the Project menu, choose Build Options. Select the Compiler tab. Make sure the optimization level (Opt Level) is set to "None", and Generate Debug Info is set to "Full Symbolic Debug (-g)".
5. Click OK.
6. From the Project Configuration drop down menu, choose Debug if it is not already selected.

file:///C:/Documents and Settings/Pierre-Armand/Local Settings/Temp/~hhBA53.htm

26/09/2007



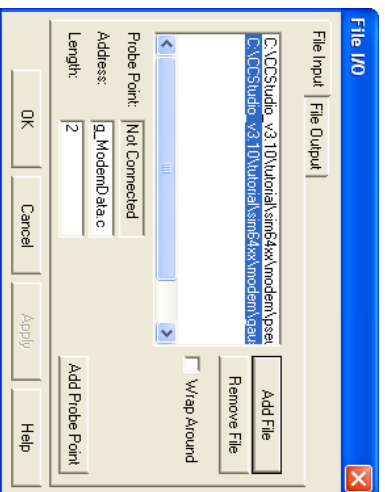
- From the Project Menu, choose Rebuild All.
  - From the File menu, choose Load Program. Browse to `C:\CCStudio_v3.10\tutorial\target\modern\Debug\Select.modem.out` and click Open.
  - From the Debug menu, choose Go Main.
  - From the Project View window, right-click on `modernx.c` and choose Open from the context menu, if it is not already open.
  - This will open `modernx.c`. Right-click in the line of the following function (line 337) in the main function. The actual line number may vary.  
`ReadNextData();`
  - From the pop-up dialog, choose Toggle Software Breakpoint. See the main help for more information on software and hardware breakpoints.
  - From the Debug menu, choose Run.
  - Right-click on the `ReadNextData();` function and choose Toggle Software Probe Point. See the main help for more information on software and hardware probe points.
  - From the File menu, select File I/O, remove any files in the File Input or File Output tabs.
  - On the File Input tab, select Add File.
  - From the File Input window, select `pseudo.dat` and click Open. You may need to navigate up one folder level to find the file.
  - In the Address field, type: `g_ModemData.dataSymbols`.
  - In the Length field, type 1.
  - Check the Wrap Around check box.
  - On the File Input tab, select Add File.
  - From the File Input tab, select `gauss32.dat` and click Open.
  - In the Address field, type: `g_ModemData.cNoise`.
  - In the Length field, type 2.
- Your File I/O dialog should resemble this:

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007

Getting Started With the Code Composer Studio Tutorial

Page 82 sur 548



Now let's connect a Probe Point to the data file.

- In the File I/O dialog, select `gauss32.dat` and click Add Probe Point.
- Select the Probe Point tab and click on the line that says `modernx.c` line 337 (0x00001880) --> No Connection. The actual line number may vary.
- In the Connect to list, select FILE IN: `gauss32.dat`.
- Select Replace.
- Click OK.
- In the File I/O dialog, select `pseudo.dat`, and click Add Probe Point.
- Select the Probe Point tab and click on the line that says `modernx.c` line 337 -->. The actual line number may vary.
- In the Connect to list, select FILE IN: `pseudo.dat`.
- Select Add.
- Click OK twice to save the changes and dismiss the File I/O dialog.

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007



## Amplitude vs Time Diagram

CCS - L5

Next, you will create an Amplitude vs. Time Diagram for the In-phase delay line which holds the modulation shaping filter.

1. From the View menu, choose Graph→Time/Frequency.
2. Fill in the Graph Property Dialog with the values in the following image. Please note that the Start Address property "g\_ModemDataIdelay" has a capital I, not a lower-case L. Scroll down or resize the dialog box to see all the properties.



[More about the Graph Property Dialog Input fields](#) (Click to view additional information)

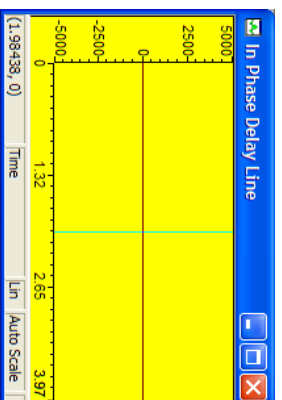
file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007

## Getting Started With the Code Composer Studio Tutorial


Page 84 sur 548

3. Click OK to save your input and create the Amplitude vs Time diagram. It should resemble this:



4. Right-click on the graph and choose Clear Display from the context menu.

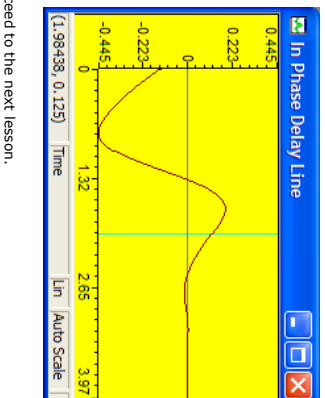
Now let's Animate the Diagram by connecting it to a data source.

1. From the Debug menu, choose Probe Points.
2. From the Probe Point list, select one of the lines that reads MODEMTX.C line 337 -->. The actual line number may vary.
3. In the Connect To field, choose In Phase Delay Line and click Add.
4. Click OK to save your input and close the dialog box.
5. From the Debug menu, choose Animate, or click on the Animate shortcut  on the tool bar.

The In Phase Delay Line Graph should now resemble this:

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007



- From the Debug menu, choose Halt to stop debug and proceed to the next lesson.



## Eye Diagram

CCS - L5

In this part of the lesson, you will create an Eye Diagram.

- From the View menu, choose Graph->Eye Diagram.
- Fill in the Graph Property Dialog with the values in the following image. Scroll down or resize the dialog box to see all the properties.

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007

Getting Started With the Code Composer Studio Tutorial

Page 86 sur 548

Graph Property Dialog	
Display Type	Eye Diagram
Graph Title	Eye Diagram
Trigger Source	Yes
Interleaved Data Sources	No
Start Address - Data Source	q.ModemData.Delay
Start Address - Trigger Source	q.ModemData.SymbolClock
Acquisition Buffer Size	32
Index Increment	1
Persistence Size	6400
Display Length	32
Minimum Interval Between Triggers	32
Pre-Trigger (in samples)	16
DSP Data Type	32-bit signed integer
Q-value	15
Sampling Rate (Hz)	64000
Trigger Level	0
Maximum Y-value	.75
Axes Display	On
Time Display Unit	ms
Status Bar Display	On
Grid Style	Zero Line
Cursor Mode	Data Cursor

OK Cancel Help


- Click OK to save your input and create the Eye diagram. It should resemble this:

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

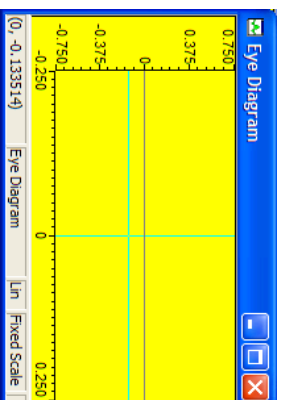
26/09/2007

4. Right-click on the graph and choose Clear Display from the context menu.

Now let's Animate the Eye Diagram by connecting it to a data source.

1. From the Debug menu, choose Probe Points.
2. From the Probe Point list, select one of the lines that reads MODEMTX.C line 337 -->. The actual line number may vary.
3. In the Connect To field, choose Eye Diagram and click Add.
4. Click OK to save your input and close the dialog box.
5. From the Debug menu, choose Animate, or click on the Animate shortcut  on the tool bar.

The Eye Diagram should now resemble this:



file:///C:/Documents and Settings/Pierre-Armand/Local Settings/Temp/~hhBA53.htm

26/09/2007

Getting Started With the Code Composer Studio Tutorial

Page 88 sur 548

6. From the Debug menu, choose Halt to stop debug and proceed to the next lesson.

[More about this Eye Diagram](#) (Click to view additional information)

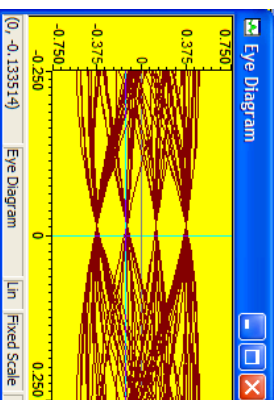


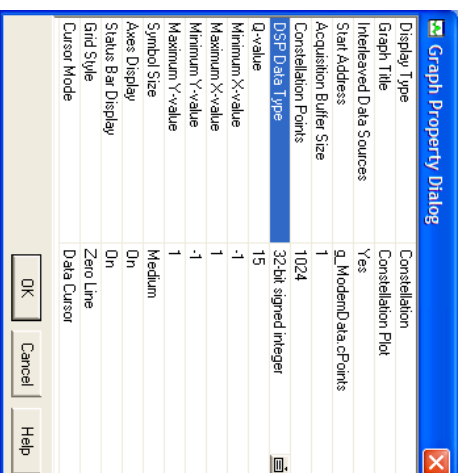
### Constellation Diagram

CCS - L5

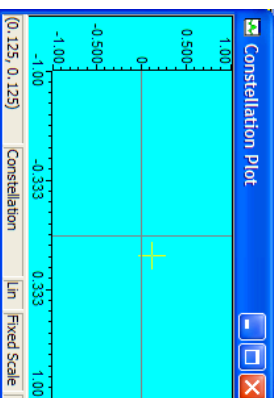
In this part of the lesson, you will create a Constellation Plot Diagram.

1. From the View menu, choose Graph-->Constellation.
2. Fill in the Graph Property Dialog with the values in the following image. Scroll down or resize the dialog box to see all the properties.





- Click OK to save your input and create the Constellation Plot diagram. It should resemble this:



- Right-click on the graph and choose Clear Display from the context menu.


Now let's Animate the Constellation Plot by connecting it to a data source.

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

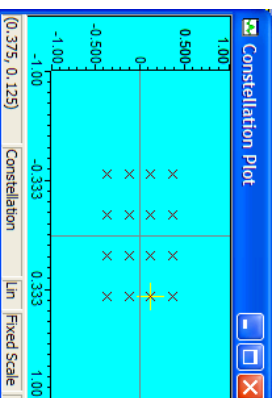
26/09/2007

Getting Started With the Code Composer Studio Tutorial

Page 90 sur 548

- From the Debug menu, choose Probe Points.
- From the Probe Point list, select one of the lines that reads MODEMTX.C line 337 -->. The actual line number may vary.
- In the Connect To field, choose Constellation Plot and click Add.
- Click OK to save your input and close the dialog box.
- From the Debug menu, choose Animate, or click on the Animate shortcut  on the tool bar.

The Constellation Plot should now look something like this:



- From the Debug menu, choose Halt to stop debug and proceed to the next lesson.

The constellation plot is one of the most effective measures of signal noise detection. As such, it plays an important role in telecommunications applications of DSP code in general and is used as a noise detector for our modem application. The input signal is separated into two components and the resulting data is then plotted with time using the Cartesian coordinate system. Toggling Interleaved Data Sources to "On" allows us to make this Cartesian "separation" possible. System noise perturbations can be seen with constellation points being plotted in a non-uniform manner. We can thus effectively picture signal noise as the "offset" causing the expected (theoretical) message point to be different from the (actual) received signal point in the constellation plot.

In this case, you are plotting the constellation points that are created as part of the source code – we are thus working on a per-sample basis (hence Acquisition Buffer set to 1) and are creating a plot using a maximum of 1024 constellation points (Constellation Points setting).



file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

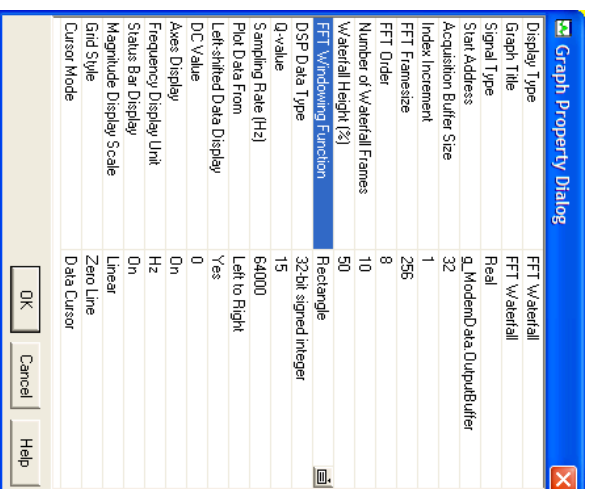
26/09/2007

**Fast Fourier Transform (FFT) Diagram**

CCS - L5

In this part of the lesson, you will create a Fast Fourier Transform (FFT) Diagram.

1. From the View menu, choose Graph→Time/Frequency.
2. Fill in the Graph Property Dialog with the values in the following image. Scroll down or resize the dialog box to see all the properties. On certain devices, a row for Maximum Y-value may appear, use the default value for this row.



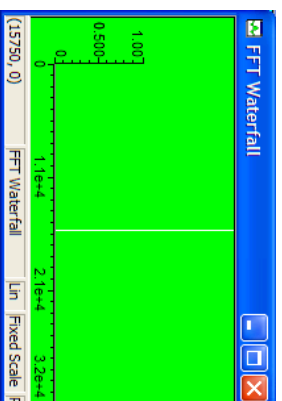
3. Click OK to save your input and create the FFT diagram. It should resemble this:

file:///C:/Documents and Settings/Pierre-Armand/Local Settings/Temp/~hhBA53.htm

26/09/2007


**Getting Started With the Code Composer Studio Tutorial**

Page 92 sur 548



4. Right-click on the graph and choose Clear Display from the context menu.

Now let's Animate the FFT Waterfall by connecting it to a data source.

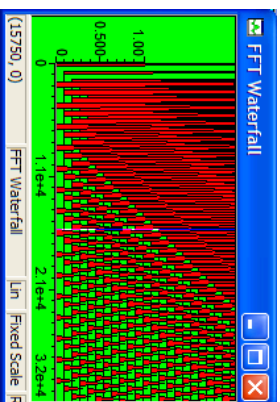
1. From the Debug menu, choose Probe Points.
2. From the Probe Point list, select one of the lines that reads MODEMTX.C line 337 -->. The actual line number may vary.
3. In the Connect To field, choose FFT Waterfall and click Add.
4. Click OK to save your input and close the dialog box.
5. From the Debug menu, choose Animate, or click on the Animate shortcut  on the tool bar.

The FFT Waterfall should now resemble this:

file:///C:/Documents and Settings/Pierre-Armand/Local Settings/Temp/~hhBA53.htm

26/09/2007

6. From the Debug menu, choose Halt to stop debug.



FFT Waterfall is a magnitude vs. frequency plot of the signal, formed by performing an FFT (Fast Fourier Transform) on the data contained in the Display Buffer, and then plotted as a frame. A chronological series of these frames forms an FFT waterfall graph. For more information on FFT and its implementation, please refer to any text covering DSP theory. In this case, we are plotting the FFT with respect to time of the `g_ModemData.OutputBuffer` variable, based on settings which include the following:

- FFT Framesize – specifies the number of samples used in each FFT computation
- FFT Order – specifies FFT size =  $2^{\text{FFT Order}}$ , with zero-padding if  $\text{FFT Framesize} < \text{FFT Order}$
- Number of waterfall frames – specifies the number of waterfall frames to be displayed
- Waterfall Height - percentage of vertical window height used to display a waterfall frame
- FFT Windowing Function – windowing function (used to limit the extent of the sequence to be transformed via FFT so that the frequency characteristics are reasonably stationary over the duration of the window)

This concludes the Data Visualization Lesson. Please close all graphs and remove all probe points and breakpoints before moving to the next lesson.



## Profiling Code Execution

CCS - L6

In this lesson, you will learn how to profile an application using Code Composer Studio tools.

**Note:** This tutorial was created for use with the CGX, C55x, C2x, and ARM simulators. The C54x simulators may not have the profile collector infrastructure deployed on them, and might

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007

## Getting Started With the Code Composer Studio Tutorial

Page 94 sur 548

[show no activities for profiling.](#)

Learning Objectives:

- Prepare a project for Profiling
- Use the Function utility to quickly profile functions
- Generate a profiling report
- Configure the profiler clock

Examples used in this lesson: modern

Target Configuration: This lesson was written for use with a simulator, and will not function properly with a DSK hardware configuration. For more information on hardware profiling limitations in Code Composer Studio 3.1, please see the online help .

Application Objective:

This lesson utilizes the Code Composer Studio profiler to demonstrate how to identify sections of code that require optimization. You will learn how to profile C functions using function ranges and addresses. You will also generate reports on the profiling data, and configure the clock.

This lesson consists of the following steps. To go directly to a step, click on the link below:

[Preparing the Application](#)

[Profiling Functions](#)

[Defining a Range](#)

[Dragging Function Ranges](#)

[Report Generation](#)

[Configuring the Clock](#)

[Profiling Tips](#)



The forward arrow will take you to the next page in this lesson.

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007

## Preparing the Application

CCS - L6

In this part of the lesson, you will prepare the Modem example for profiling and create the profile session name.

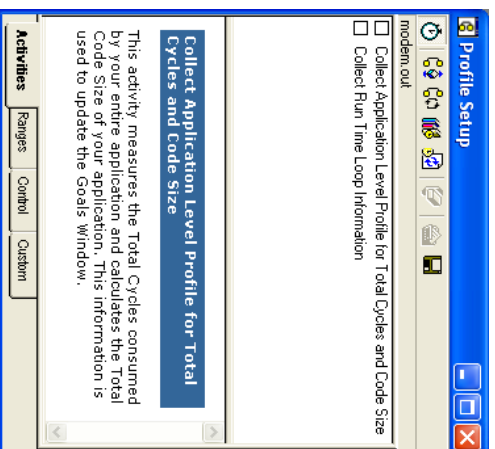
1. From the Project menu, choose Open.
2. Browse to [C:\CCStudio\\_v3.10\tutorial\target\modem\](C:\CCStudio_v3.10\tutorial\target\modem\).
3. Select `modem.pjt` and click Open.
4. From the Project menu, choose Build Options. Select the Compiler tab. Make sure the optimization level (Opt Level) is set to "None", and Generate Debug Info is set to "Full Symbolic Debug (-g)".
5. Click OK.
6. From the Project Configuration drop down menu, choose Debug if it is not already selected.
7. From the Project Menu, choose Rebuild All.
8. From the File menu, choose Load Program.
9. Browse to [C:\CCStudio\\_v3.10\tutorial\target\modem\Debug\](C:\CCStudio_v3.10\tutorial\target\modem\Debug\).
10. Select `modem.out` and click Open.
11. From the Profile menu, choose Setup. The Profile Setup window will appear. Your Activity list may vary.


<file:///C:/Documents and Settings/Pierre-Armand/Local Settings/Temp/~hhBA53.htm>

26/09/2007

## Getting Started With the Code Composer Studio Tutorial

Page 96 sur 548



12. Check the icons at the top of the Profile Setup tool to determine if profiling is enabled. The Enable/Disable Profiling icon  must be toggled on to enable profiling. Profiling may not be enabled if the program has started running or if it was explicitly disabled. Data will not be collected if profiling is not enabled.
13. Choose the Custom tab and select the **CPU cycles** check box (or cycle-Total for certain devices.)

Now that you have a profile session ready, in the next exercise you will learn how to quickly profile all or some of the functions in a project.



## Profiling Functions

CCS - L6

Here you will be introduced to methods used to profile C functions.

1. In the Profile Setup window, click on the Ranges tab.

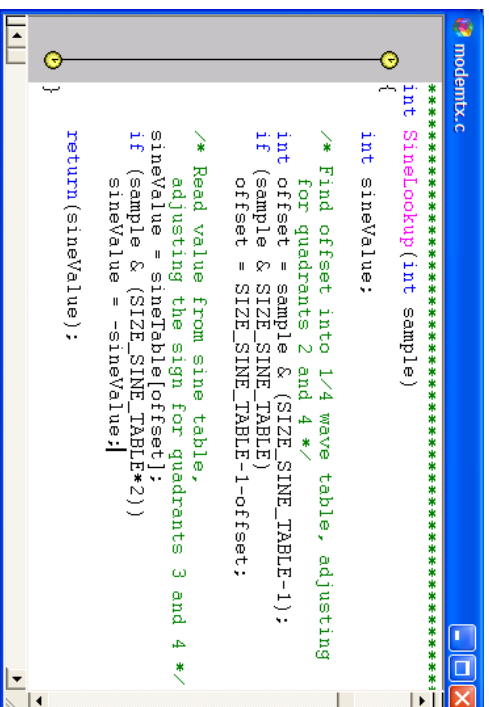
<file:///C:/Documents and Settings/Pierre-Armand/Local Settings/Temp/~hhBA53.htm>

26/09/2007



2. Navigate to the Project View window and open the modem.pjt project.
3. Click on the Source folder, and open modemx.c by double clicking modemx.c.
4. Highlight the lines containing, [int SineLookup\(int sample\)](#) (line 53-70), and drag it to the Functions line in the Profile Setup Ranges tab under modem.out. The actual line number may vary.
5. Expand the view of profiled functions in modem.out by clicking the + "plus" sign next to Functions and Enabled in the profile setup window.

There is now an entry in the profile window for SineLookup. In modemx.c, the selection margin in the editor has also highlighted the range of the function, as in the following example.



```

modemx.c
*****
int SineLookup(int sample)
{
    int sineValue;



    /* Find offset into 1/4 wave table, adjusting
       for quadrants 2 and 4 */
    int offset = sample & (SIZE_SINE_TABLE-1);
    if (sample & SIZE_SINE_TABLE)
        offset = SIZE_SINE_TABLE-1-offset;

    /* Read value from sine table,
       adjusting the sign for quadrants 3 and 4 */
    sineValue = sineTable[offset];
    if (sample & (SIZE_SINE_TABLE*2))
        sineValue = -sineValue;

    return (sineValue);
}

```

You can also import a function by right-clicking on the Ranges window, and choosing Create Profile Item.

6. To quickly profile all functions in the program, click the Enable/Disable All Functions icon  on the top toolbar of the Profile Setup window.
7. From the Profile menu, choose Viewer. The Profile Viewer has a default set of columns. To add or hide columns, choose the column editing  icon.

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

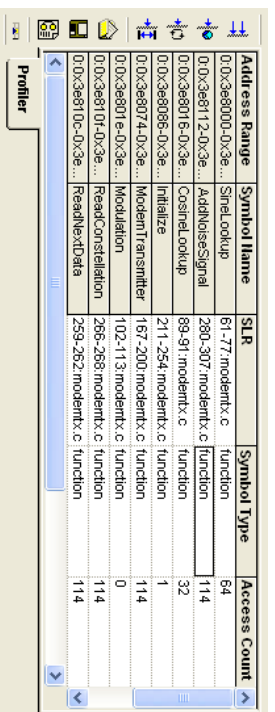
26/09/2007

Getting Started With the Code Composer Studio Tutorial

Page 98 sur 548

8. From the Debug menu, choose Run, and let the program run for about one minute.
9. Halt the program by choosing Debug→Halt. You may need to click on Debug→Reset CPU to see results.

Now you can see the results of your profiling in the Profile Viewer window, as in the following example. Your numbers may vary.



Address Range	Symbol Name	SLR	Symbol Type	Access Count
010X3e8000-DX3e...	SineLookup	61-77	function	64
010X3e8112-DX3e...	AddBusSignal	280-307	function	114
010X3e8016-DX3e...	CosSineLookup	89-91	function	32
010X3e8086-DX3e...	Initialize	211-254	function	1
010X3e8074-DX3e...	ModemTransmitter	167-200	function	114
010X3e801e-DX3e...	Modulation	102-113	function	0
010X3e810f-DX3e...	ReconfConstellation	268-268	function	114
010X3e810c-DX3e...	ReconfConstellation	259-262	function	114

In the following exercise, you will learn how to profile a range of functions.



### Defining a Range

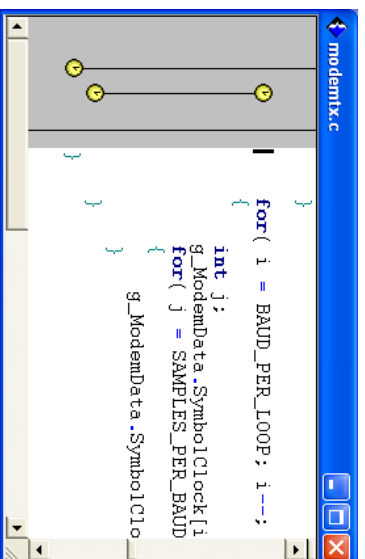
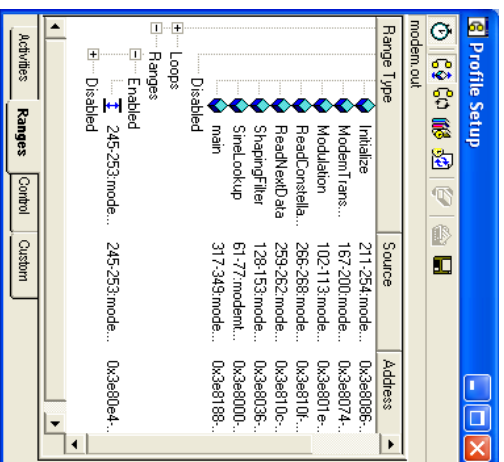
CCS - L6

In this exercise, you will be introduced to methods used to profile ranges of lines and addresses.

1. Select the Ranges tab in the Profile Setup window.
2. Highlight the last "Top" loop under the Initialize function on lines 245-253, and drag it into Ranges line of the Ranges tab. The actual line number may vary. You can also right-click on it and choose Profile→Range. The line numbers will appear under Ranges in the Profile Setup window. Your window display may vary.

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007



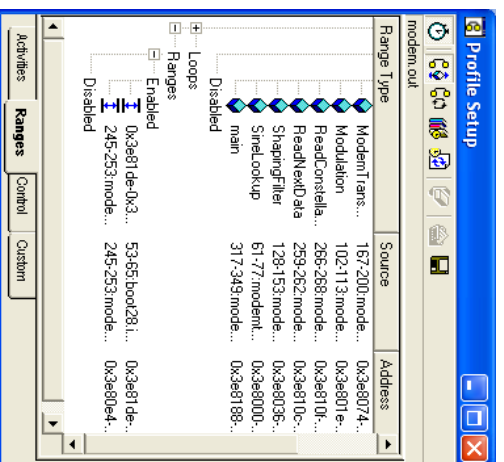
file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007

Getting Started With the Code Composer Studio Tutorial

Page 100 sur 548

- From the View menu, choose Disassembly to open the disassembly window.
- Right click in the Disassembly window and select Start Address.
- Enter c\_in00 as the starting address.
- In the Disassembly window, select the first 4 lines under c\_in00 and drag them on to the Ranges list in the Profile Setup window. Your window display may vary.



- From the Debug menu, choose Reset CPU.
- From the File menu, choose Reload Program.
- From the Debug menu, choose Run. Let the application run for about one minute.
- From the Debug menu, choose Halt. You may need to click on Debug→Reset CPU to see results.
- From the Profile menu, choose Viewer, if it is not already open. Now you can see the profiling results.

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007

In the following exercise, you will learn how to use halt and resume collection points to profile.



### Dragging Function Ranges

CCS - L6

In this exercise, you will learn how to use halt and resume points with the Code Composer Studio profiler. First, you need to clear any previous profiling settings:

1. In the Profile Setup window, select the Ranges tab.
2. Select any ranges and hit the Delete key, and toggle the Enable/Disable All Functions button off, select any enabled functions and hit the space key to disable them.
3. Make modemtx.c the active window by double clicking on it in the Project View window or selecting it in the main workspace.
4. While the Ranges tab is active, right click on the open bracket after the ShapingFilter function in modemtx.c (line 121) and choose Profile--Range. The actual line number may vary.  
The for loop in the ShapingFilter function is already optimized. Therefore, we will tell the Profiler to ignore it.
5. Select the Control tab in the Profile Setup window.
6. Highlight the first line of the first **"for"** loop under the Shaping Filter function (line 125) and drag it onto the text that says "Halt Collection" in the Profile setup window. The actual line number may vary.
7. Highlight the first line of the third **"for"** loop under the Shaping Filter function (line 137) and drag it onto the text that says "Resume Collection" in the Profile Setup window. The actual line number may vary.

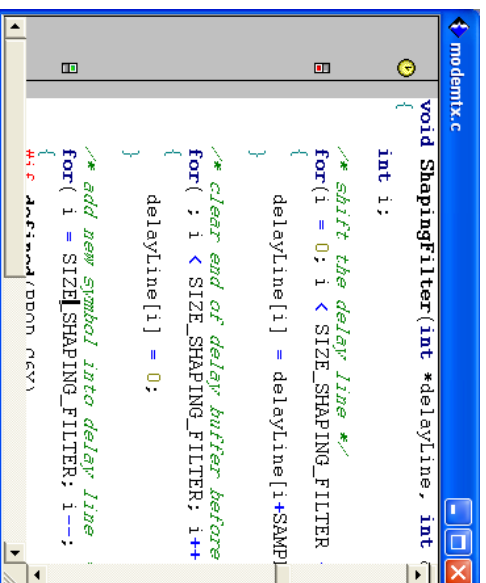
You should see the following items displayed in the selection margin of modemtx.c:

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007

Getting Started With the Code Composer Studio Tutorial

Page 102 sur 548

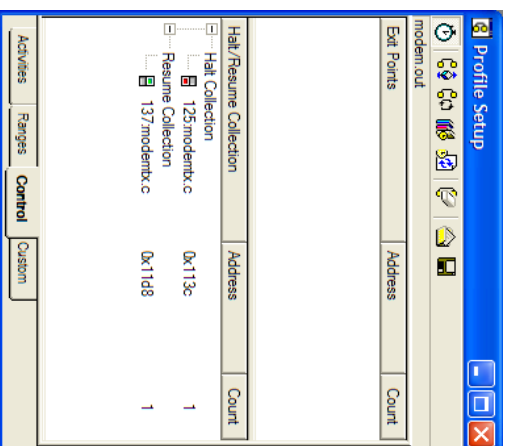


```
modemtx.c
{
void ShapingFilter(int *delayLine, int
    int i;
    /* shift the delay line */
    for(i = 0; i < SIZE_SHAPING_FILTER
    {
        delayLine[i] = delayLine[i+SAMP
    }
    /* clear end of delay buffer before
    for( ; i < SIZE_SHAPING_FILTER; i++
    {
        delayLine[i] = 0;
    }
    /* add new symbol into delay line
    for( i = SIZE_SHAPING_FILTER; i--;
    {
        /* ... */
    }
}
```

The Profile Setup Control tab should now resemble this:

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007



Now, when you profile the function, it will ignore the for loop. You can also create halt, exit, and resume points by right-clicking on the Profile Setup window under the Control tab and choosing Add Control Point.

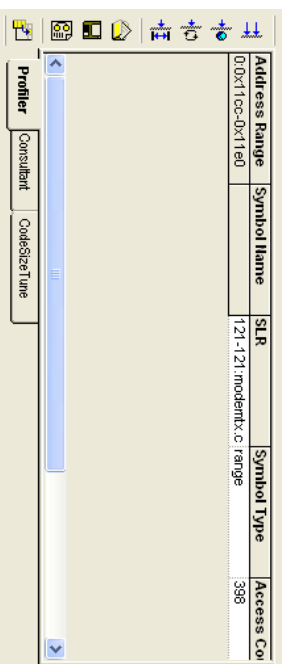
8. From the Debug menu, choose Reset CPU.
9. From the File menu, choose Reload Program.
10. From the Debug menu, choose Run. Let the program run for 15-30 seconds.
11. From the Debug menu, choose Halt. You may need to click on Debug-Reset CPU to see results. Now you can view your profile results by clicking on the Profile Viewer. Your numbers may vary.

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007

Getting Started With the Code Composer Studio Tutorial

Page 104 sur 548



In the next exercise, you will learn how to generate a report.



### Report Generation

CCS - L6

Here, you will learn how to generate a report of profiling data. To generate a report using data stored in the profiler:

1. Click on the Export data to a text delimited file icon  on the left hand side of the **Profile Viewer** window.
2. Enter the name and path of the file that you wish to create in the report generation dialog. By default, a text file with the same name as your profile session is created in your project directory.

This creates a tab delimited file that you can import into a spreadsheet application for further analysis.

In the next exercise you will learn how to configure the clock.



### Configuring the Clock

CCS - L6

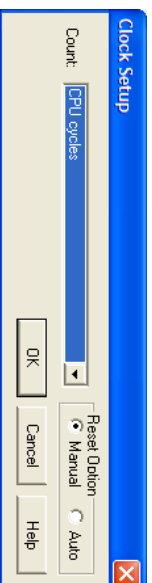
file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007

We will now configure the clock used by the profiler.

1. From the Profile menu, choose Clock, then Setup. If the Clock Setup option is not available, un-check the cycle box under the Custom tab of the Profile Setup window.
2. From the drop down box labelled Count, select an Event.

You can select from a variety of events. The number and type of events available depends on your connected simulator or emulator.



3. Click OK to close the dialog and save the changes.

For more information on Instruction cycle time and Pipeline adjustments, see the online help.

This concludes the main Profiling Code Execution lesson. For additional tips on Profiling, please see the [next lesson](#).



### Profiling Tips

---

CCS - L6

This section highlights some other things you might want to try out with the profiler:

1. You can arrange the order of columns in the Profile Viewer so the ones you use the most are at the beginning. See the [Tuning Dashboard](#) tutorial for more information on the Profile Viewer.
2. You can drag functions from the Symbol Browser into Profile Setup to quickly profile a function. See the [Tuning Dashboard](#) tutorial for more information on the Profile Setup window.
3. It is possible to use profiling data to optimize your build options using the Code Size Tune tool. See the [CodeSizeTune](#) tutorial for more information.

For more information, consult the appropriate CCS online help section.

This concludes the Profiling Code Execution lesson.

file:///C:/Documents and Settings/Pierre-Armand/Local Settings/Temp/~hhBA53.htm

26/09/2007

Getting Started With the Code Composer Studio Tutorial

Page 106 sur 548



### Using GEL Language

---

CCS - L7

GEL (General Extension Language) is an interpretive language that allows you to write functions to configure the Code Composer Studio™ IDE as well as access the target processor or simulator. It allows you to automate tasks similar to macros, however, its C-like syntax allows it to be much more powerful. GEL supports basic C constructs such as while loops and if structures. In addition, GEL has a rich library of functions at your disposal.

#### Learning Objectives:

- Use GEL scripts to create GUI objects
- Control various DSP target components using GUI targets
- Define local variables for the GEL application
- Utilize GEL-created custom controls to investigate the effects of particular DSP variable changes on algorithm execution

Examples used in this lesson: gelsolid

**Target Configuration:** This tutorial was developed with Code Composer Studio™ 3.00, which uses version 5.00 of the Compiler, and should be executed either on the CG416 Device Cycle Accurate Simulator, Little Endian, or the CG211 Device Cycle Accurate Simulator, Little Endian. Executing the tutorial on different execution Platforms may require modification of the linker command file due to differences in the memory map. For more information on the Setup utility, access the [Configuring Target Devices](#) tutorial or the online help.

#### Application Objective:

In this lesson you will learn how to use GEL functions to automate common tasks such as compiling, linking and running your DSP application. You will also learn how to write completely autonomous GEL functions that declare all variables within the function.

This lesson consists of the following steps. To go directly to a step, click on the link below:

- [Writing Your First GEL Application](#)
- [Defining Local Variables](#)
- [Using GEL Automation](#)

file:///C:/Documents and Settings/Pierre-Armand/Local Settings/Temp/~hhBA53.htm

26/09/2007



The forward arrow will take you to the next page in this lesson.

### Writing Your First GEL Application

CCS - L7

In this section you will learn how to write and execute a simple GEL function. You can learn more about GEL by consulting the online help.

1. From the File menu, choose **New→Source File**.
2. In the source file, type the following text:

```
menuItemem "GEL_Welcome_Tool*";
hotMenuItem Welcome_To_GEL_Function()
{
    GEL_Textout("GEL is a solid tool.\n");
}
```

3. From the File menu, choose **Save As**.
4. Browse to **C:\CCStudio\_v3.10\tutorial\target\gelsolid\**.
5. In the **Save as type box**, choose **General Extension Language Files (\*.gel)**.
6. Type `test.gel` as the file name and click **Save**.

Next, let's load the GEL file and see how easy it is to use GEL.

7. From the File menu, choose **Load GEL**.
8. Browse to the location where you saved `test.gel`.
9. Select `test.gel` and click **Open**.
10. From the GEL menu, choose **GEL Welcome Tool→Welcome\_To\_GEL\_Function**. The following text appears in the output window:

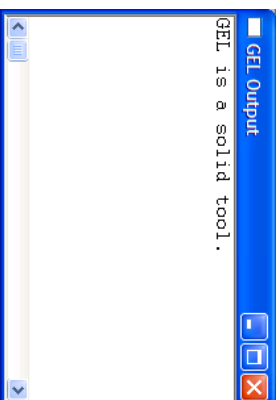
file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007

### Getting Started With the Code Composer Studio Tutorial

Page 108 sur 548

In the following exercise, you will use local variables to display output multiple times.



### Defining Local Variables

CCS - L7

In this section you will learn how to use local variables in GEL. Essentially, local variables can be used in the same manner as in standard C; they must be declared together in an implicit 'data section' and used below in the implicit 'program section'. The simple GEL function you will develop prints a message on the screen 10 times.

1. From the File menu, choose **New→Source File**.
2. In the source file, type the following text:

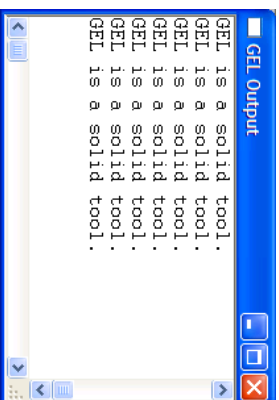
```
menuItemem "GEL_Welcome_Tool*";
hotMenuItem Using_Local_Variables()
{
    int i;
    for (i = 0; i < 10; i++)
        GEL_Textout("GEL is a solid tool.\n");
}
```

3. From the File menu, choose **Save As**.
4. Browse to **C:\CCStudio\_v3.10\tutorial\target\gelsolid\**.
5. In the **Save as type box**, choose **General Extension Language Files (\*.gel)**.

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007

6. Type *localvariables.gel* as the file name and click Save. Click OK to replace the file, if necessary. Next, let's load the GEL file and see how easy it is to use GEL local variables in GEL.
7. From the File menu, choose Load GEL.
8. Browse to the location where you saved *localvariables.gel*.
9. Select *localvariables.gel* and click Open.
10. From the GEL menu, choose GEL Welcome Tool→Using\_Local\_Variables. The following text appears in the output window (and should print 10 times):



In the following exercise you will use GEL to automate common tasks using GEL library functions.



### Using GEL Automation

CCS - L7

In this section, you will learn how to automate common tasks by using GEL to open, build, and run an existing project. We will be using pre-existing standard GEL library functions.

1. From the File menu, choose Open.
2. Browse to *C:\CCStudio\_v3.10\tutorial\target\gelsolid\*.

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007

### Getting Started With the Code Composer Studio Tutorial

Page 110 sur 548

3. To view GEL files, choose *gel* as the extension in the drop down menu. Select *projectmanagement.gel* and click Open. The gel file should appear. Select the following link to display the contents of *projectmanagement.gel*. Note the location of the purple-colored code.  
[projectmanagement.gel](#) (Click to view code)

**Note:** When specifying paths in GEL, make sure you use the double slash.

Next, you need to modify the GEL *ProjectLoad()* and GEL *Load()* functions to use the project for which you have configured CCS. The portion of the GEL function you need to modify is colored purple in the example above.

4. In the source file, select the following text:

```
GEL_ProjectLoad("C:\\Source\\hellodsp.pjt");
```

Type the following, replacing "target" with your appropriate **target** folder name:

```
GEL_ProjectLoad("C:\\CCStudio_v3.10\tutorial\target\gelsolid\hellodsp.pjt");
```

5. In the source file, select the following text:

```
GEL_Load("C:\\Source\\debug\\hellodsp.out");
```

Type the following, replacing "target" with your appropriate **target** folder name:

```
GEL_Load("C:\\CCStudio_v3.10\tutorial\target\gelsolid\debug\hellodsp.out");
```

**Caution:** After modifying a previously loaded GEL file, you must re-load the GEL file for your changes to take effect.

6. From the File menu, choose Save As.
  7. Browse to *C:\CCStudio\_v3.10\tutorial\target\gelsolid\*.
  8. In the Save As type box, choose **General Extension Language Files (\*.gel)**.
  9. Type *projectmanagement.gel* as the file name and click Save. Click OK to replace the file, if necessary.
- Next, let's load the GEL file and see how easy it is to automate tasks using GEL.

10. From the File menu, choose Load GEL.
11. Browse to the location where you saved *projectmanagement.gel*.

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

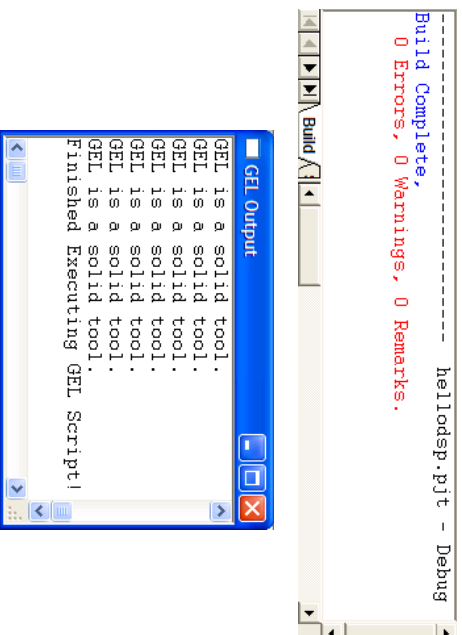
26/09/2007

12. Select projectmanagement.gel and click Open.

13. From the GEL menu, choose Project Management Tool->Run\_Project. This action will load, build and run the project. You may receive an error message if the program runs too quickly. This message can be safely ignored, as the GEL script will correctly execute after a few moments.

**Note:** If using a DSK, check that it is connected and halted.

You should also see the following output (Project Build output and GEL output):



14. From the Debug menu, choose Halt, if necessary.

In the following exercise, you will control DSP variables using GEL.



### Controlling DSP Variables With GEL

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007

Getting Started With the Code Composer Studio Tutorial

Page 112 sur 548

CCS - L7

In this section you will learn how to control DSP variables dynamically using GEL components. First, you will create a GEL file defining a slider control, and a dialog box to control the counterValue variable.

1. From the File menu, choose New->Source File.
2. In the source file, type the following text:
 

```
menuItem "Set Counter Value"
  dialog Set_Counter(counterParam "Load")
  {
    counterValue = counterParam;
  }
menuItem "Vary Counter Value"
  slider Vary_Counter(0, 10, 1, 1, counterParam)
  {
    counterValue = counterParam;
  }
```
3. From the File menu, choose Save As.
4. Browse to C:\CCStudio\_v3.10\tutorial\target\gelsolid\.
5. In the Save As box, choose General Extension Language Files (\*.gel).
6. Type counter.gel as the file name and click Save.

This file defines two GEL components: a dialog box and a slide control. Let's examine the syntax of each component definition.

The dialog component is created using the keyword dialog. This component takes a parameter (specified by the "load" keyword). The body of the source code assigns the parameter value counterParam to the counterValue variable contained in the DSP source code. It is important that the variable specified in the GEL file matches the one contained in the DSP application.

The slider component is created using the keyword slider. The parameters required by the slider (from left to right) are: slider minimum value, slider maximum value, the change value caused by an up or down click respectively, and finally, the parameter to which the slider value is assigned. The body of the source code assigns the parameter value counterParam to the counterValue variable contained in the DSP source code. Again, the variable specified in the GEL file must match the one contained in the DSP application.

7. From the File menu, choose Load GEL.
8. Browse to the location where you saved counter.gel.
9. Select counter.gel and click Open.

### Running the DSP Application

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007



Before you run the DSP application, you should create a breakpoint. A breakpoint will shift focus from the GEL component to the DSP application to the GEL component. The temporary shift in focus allows the GEL component to execute properly. (If you're curious about what happens without the Breakpoint - try it!)

1. In the Project View window, double-click on multispwelcome.c to open it.
2. From the File menu, choose Load Program.
3. Browse to C:\CCStudio\_v3.10\tutorial\target\gelsolid\Debug\.
4. Select hellodsp.out and click Open.
5. Right-click on line 14 (**while (TRUE)** ) and choose Toggle Software Breakpoint. Actual line numbers may vary with target. See the main help for more information on software and hardware breakpoints.
6. From the Debug menu, choose Run.
7. When the application halts at the breakpoint, choose GEL→Set Counter Value→Set\_Counter.
8. A dialog such as the one below will appear. Enter a value and press Execute. Your dialog should not close.



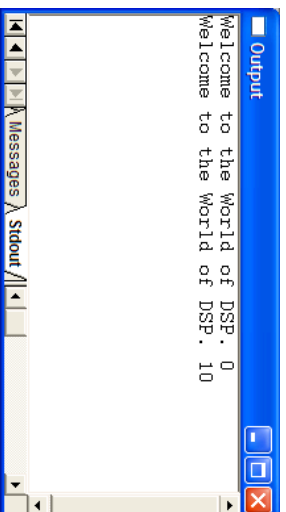
9. Enter an additional value and press Execute, then Done to close the dialog box.
10. From the Debug menu, choose Run. If you used a load value of 10, you would see the following output in the stdout window:

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

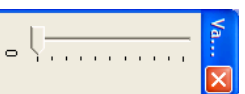
26/09/2007

Getting Started With the Code Composer Studio Tutorial

Page 114 sur 548



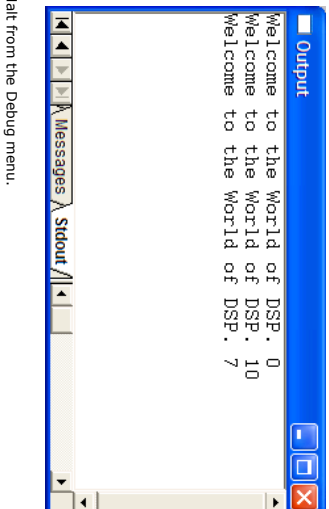
11. When the application halts at the breakpoint again, choose GEL→Vary Counter Value→Vary\_Counter. A slider control such as the one below will appear. If the application does not halt automatically, choose Halt from the Debug menu.



12. Adjust the slider up or down to change the variable.
13. From the Debug menu, choose Run to continue program execution. Notice the value of counterValue has changed.

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007



14. If the application does not halt automatically, choose Halt from the Debug menu.

This concludes the GEL tutorial. For more information about GEL functionality, see the online help.



## Configuring Target Devices

CCS - L8

In this lesson, you will learn how to manage target DSP devices.

Learning Objectives:

- Use setup to configure target devices
- Import system configurations
- Use GEL language for initialization

Examples used in this lesson: None

Application Objective:

This lesson demonstrates the advanced configuration features of the Setup program, such as simulators and device drivers. It also introduces you to different procedures for multiple processor configuration, including initialization and BYPASS devices. You will add and remove TI boards and simulators using Code Composer Studio™ Setup. You will learn how to import and export new boards and system settings as well as how to use GEL files to perform board initialization tasks.

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007

## Getting Started With the Code Composer Studio Tutorial

Page 116 sur 548

This lesson consists of the following steps. To go directly to a step, click on the link below:

- [Launching Setup](#)
- [Using Board and Simulator Defaults](#)
- [Working With Device Drivers](#)
- [Working With System Configurations](#)
- [Adding Multiple Processors](#)
- [Initialization Order](#)
- [Using BYPASS Devices](#)
- [Tips on Configuring Target Devices](#)



The forward arrow will take you to the next page in this lesson.

## Launching Setup

CCS - L8

The Code Composer Studio™ IDE allows developers to write DSP applications over a variety of TI platforms. To develop any DSP application you must identify your target hardware platform. The Setup application allows you to do just this. You can select target boards and simulators, as well as import and export new boards and simulators.

There are four situations in which the Setup utility will launch.

- If you try to run Code Composer Studio™ without specifying a target hardware platform (either board or simulator), CCSStudio will display a dialog box which will start Setup when you click Yes
- You can manually launch Setup by clicking on the Setup CCSStudio v3.1 icon on your desktop
- You can manually launch Setup from within the Code Composer Studio™ IDE by choosing Launch Setup from the File menu

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007

**Launching Via Desktop Icon**

After installation, an icon such as the following will appear on your desktop, labelled Setup CCSstudio v3.1.



Double-click on the icon to launch Setup.

**Launching from within Code Composer Studio**

From the File menu, choose Launch Setup. The Setup application will load. You can select an available configuration and select Save, then select Exit. If you have multiple configurations of the Code Composer Studio™ IDE available, you can import more than one board or simulator.

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007

**Getting Started With the Code Composer Studio Tutorial**

Page 118 sur 548

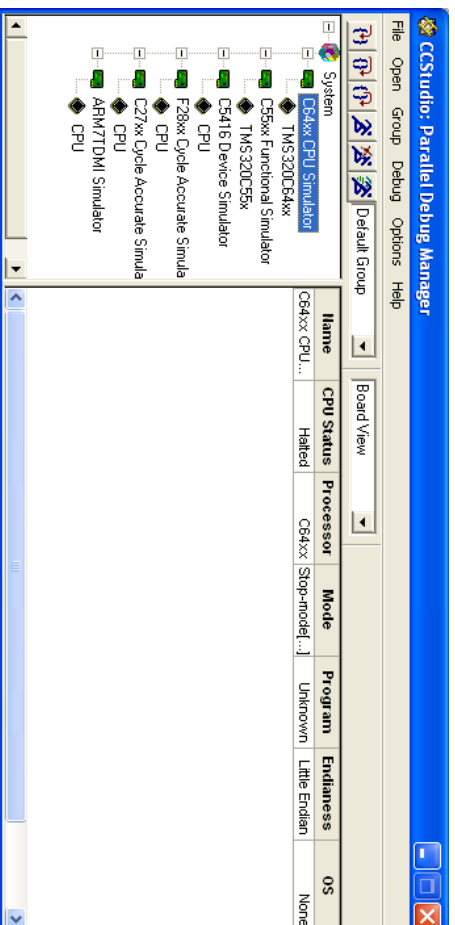
**Launching from Parallel Debug Manager**

From the File menu, choose Launch Setup. The Setup application will load. You can select an available configuration and select Save, then select Exit. If you have multiple

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007

configurations of the Code Composer Studio™ IDE available, you can import more than one board or simulator, then launch specific groups or configurations from within the Parallel Debug Manager.



In the next lesson you will examine the default settings for boards and simulators.



### Using Board and Simulator Defaults

CCS - L8

After launching Setup, notice that the middle pane of the application has a list of available boards and simulators. You can use any of these boards or simulators to run your application. You may use the boards only if you have the actual board connected to your machine.

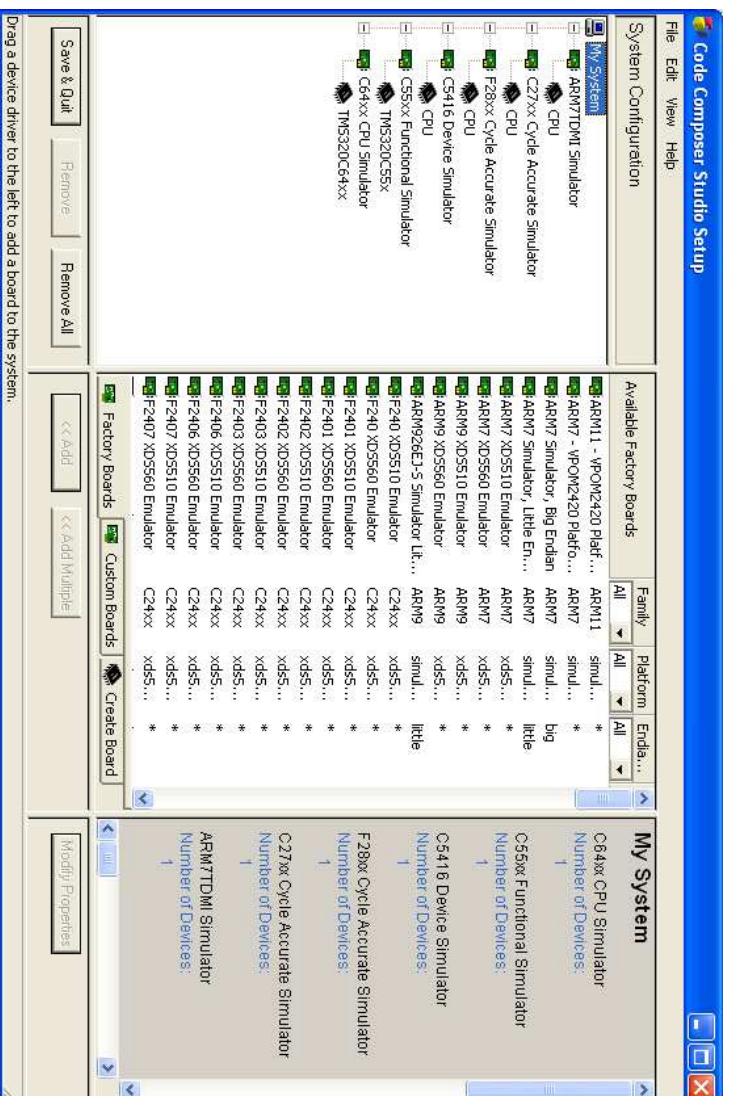
- To add a board to your system configuration, simply drag the board icon from the Available Factory Boards pane to the System Configuration pane.

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007

### Getting Started With the Code Composer Studio Tutorial

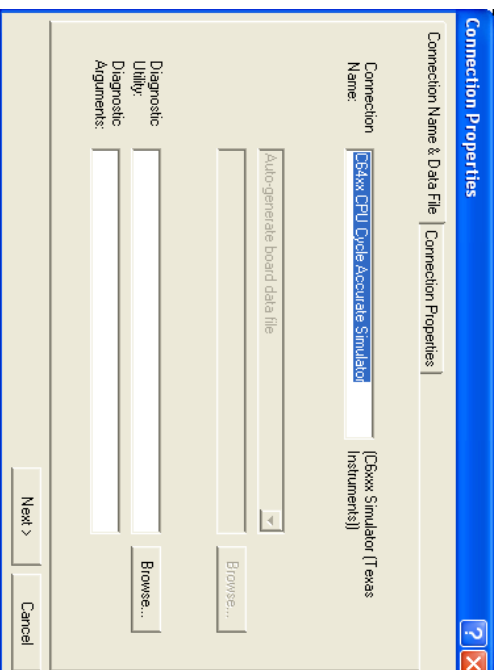
Page 120 sur 548



- To configure the board you just added, right-click on it under System Configuration, and choose Properties from the context menu.

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007



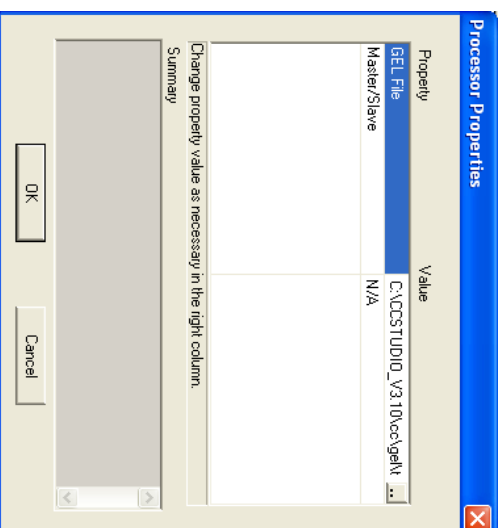
3. The Connection Name & Data File tab allows you to edit the configuration name, data file location, diagnostic utility, and diagnostic arguments.
4. Click Next to edit the Connection Properties such as the simulator configuration file, endianess, the I/O port, TCLK, or emulator name. Your options will vary depending the selected board.
5. Click Finish to save your changes and close the Properties dialog.
6. To edit processor-specific information, right-click on a processor listed under your board in the System Configuration, and select Properties from the context menu.

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007

Getting Started With the Code Composer Studio Tutorial

Page 122 sur 548



7. Code Composer Studio Setup can automatically determine the processor configuration for many target boards. You can also set your own properties, including the starting GEL file, the Master/Slave value, Startup Mode, or the BYPASS name and bit number. Other options may be available depending on your processor.
8. You can delete a board configuration from your System Configuration by simply right-clicking and selecting Remove. Alternatively, you can select the board and choose Edit-Remove, or Delete.

Now try removing the simulator or board that you previously added to your system. Familiarize yourself with both the right-clicking and menu-based method. If you have multiple configurations of the Code Composer Studio™ IDE available, you can import more than one board or simulator.

In the next lesson, you will work with the device drivers function of the Setup utility.



**Working With Device Drivers**

CCS - L8

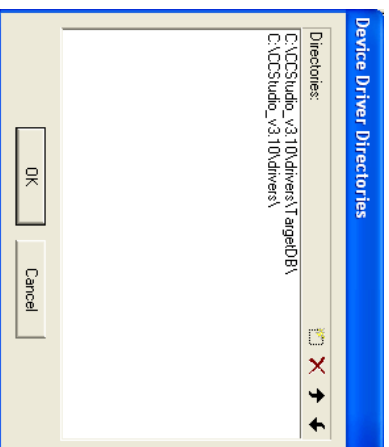
file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007

In this exercise, you will learn how to add and remove TI boards and simulators using Code Composer Studio Setup. TI simulators and boards are designed using Device Drivers. These software programs simulate the actual workings of a silicon board (memory, input/output, etc.). Code Composer Studio Setup allows you to quickly add new device drivers and remove old or obsolete device drivers.

### Installing Boards

1. Move the desired device driver and XML file to the device drivers folder in the CCSStudio install directory (C:\CCStudio\_v3.10\drivers).
2. CCSStudio will automatically pick up any new devices and add them to the device list, provided the driver is correctly configured.
3. For drivers provided by external sources, you should run the installation program supplied by the device driver provider. You may need to add an additional directory depending on how the driver is installed.
4. If the driver is installed in a different folder, that directory may be added to CCSStudio's list of driver locations.
5. Launch the Code Composer Studio Setup application.
6. Choose Edit→Device Driver Directories. The Directories dialog will appear.



7. Click on the New icon (  ) to open a new directory location.
8. Browse for the required directory and click Open.

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007

### Getting Started With the Code Composer Studio Tutorial


Page 124 sur 548

9. Click OK to save your changes and close the dialog box.

10. In the Available Factory Boards pane, select the new device driver. Information describing the board configuration that this driver supports is displayed in the Setup Commands/Information pane. Use this information to verify that you have installed the correct driver.

Caution: It is important to note that not all .drv and .dll files located on your system are valid board device drivers.

### Uninstalling Boards

1. From the Edit menu, choose Device Driver Directories.
  2. Select the desired device driver path to be removed.
  3. Click on the Delete icon (  ) to remove the path.
  4. Click OK to save your changes and close the dialog box.
- CCStudio will no longer check that path for device drivers. Removing the device driver path does not delete the driver file from your computer.

In the next lesson, you will work with the system configurations available in the Setup utility.



### Working With System Configurations

CCS - L8

In this section you will learn how to save your System Configuration pane. This is useful when you wish to change your DSP target, but would like to save your current settings for later use. You will also learn how to restore previously saved configurations.

### Exporting a Configuration

1. First, you will create a system configuration to export. To add a board to your system configuration, simply drag the board icon from the Available Factory Boards pane to the System Configuration pane.
2. Add more boards as desired.

**Note:** Multiple configurations of some boards are not supported. If the configuration you are setting up is not supported, an error message will let you know.

3. From the File menu, choose Save to save the configuration.

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

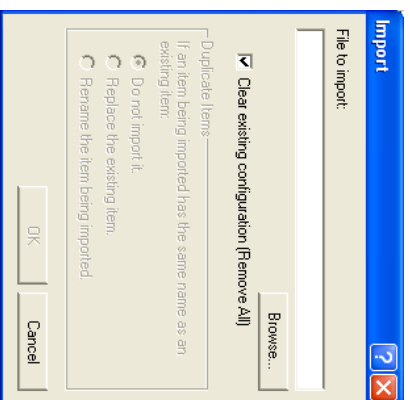
26/09/2007

4. From the File menu, choose Export.
5. Browse to `C:\CCStudio_v3.10\drivers\import\`.
6. In the File Name field, type a name for the file.
7. In the Save as type dialog, select `*.ccs` as the file type and click Save.

### Importing a Configuration

Now you can import the system configuration you just created. First, you need to clear the existing system configuration.

1. From the File menu, choose Remove All. Click Yes in the confirmation dialog.
2. From the File menu, choose Import. The **Import** dialog will appear.



3. Choose **Browse** and select the file you created in the previous section.
4. Click Open.
5. Use all the other default settings and click OK to load the selected configuration file.

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007

### Getting Started With the Code Composer Studio Tutorial

Page 126 sur 548

6. From the File menu, choose Save, and then Exit, or choose the Save & Quit button. The following dialog will appear:



7. Select Yes to start CCSStudio on exit.
8. You have successfully imported your system configuration settings. As done previously, clear your entire system configurations by selecting Remove All from the File menu in Setup. Click Yes to clear the system configuration.
9. To reload the previous system configuration, click Revert to Saved Configuration from the File menu in Setup, and click Yes to load the system configuration from the system registry.

In the next lesson you will add multiple processors using the Setup utility.



### Adding Multiple Processors

CCS - L8

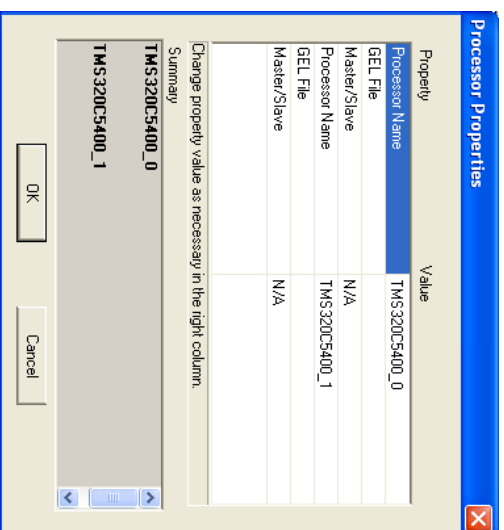
First, launch Setup by clicking on the Setup CCSStudio v3.1 icon on your desktop.

If you are using an actual hardware board (not a simulator), you may configure the Code Composer Studio™ IDE to use multiple processors.

1. To add a board to your system configuration, simply drag the board icon from the Available Factory Boards pane to the System Configuration pane. Select a non-simulator board and drag-and-drop it on the left-pane.
2. Only devices that support multiple processors will allow you to add multiple processors. Select a supported device in your System Configuration.
3. Click on the <<Add Multiple button at the bottom of the middle pane.
4. Specify the number of devices desired.
5. Click OK.
6. A dialog box will appear, allowing you to specify names for the processors, GEL files, and other processor properties. (Your device identification and values may vary.)

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007



7. Change names and values as desired, and click OK to finish. The processors will be added to your configuration.
8. The next time you run CCSStudio™, the Parallel Debug Manager will open. You can then select one or more configurations from the Open menu.

**Note:** For each configuration you select and run, another instance of CCSStudio™ will run.

In the next lesson, you will learn how to specify the initialization order for processors.

### Initialization Order

CCS - L8

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

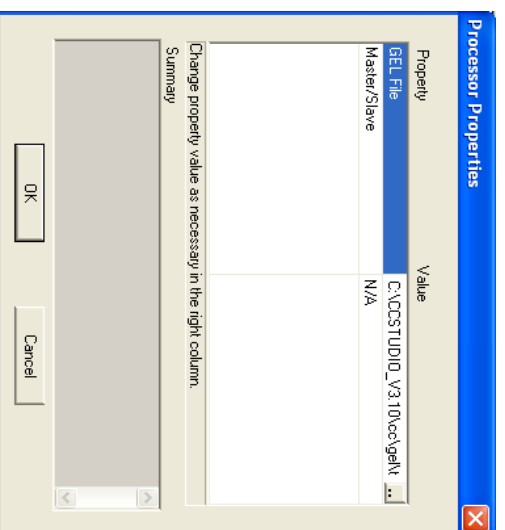
26/09/2007

### Getting Started With the Code Composer Studio Tutorial

Page 128 sur 548

In this section you will learn how to specify the initialization order for processors. This is used when a particular board has multiple processors on it. To change the initialization order of the processors, you need to configure the Master/Slave settings for each processor. If the Master/Slave property is set, CCSStudio will connect all the master devices on startup, and then the slave devices.

1. Re-open the Processor Properties dialog of the board set up previously in the lesson entitled [Adding Multiple Processors](#) (right-click on a processor and select Properties).
2. Select Master or Slave from the drop down Master/Slave property. Repeat for each processor. If the Master/Slave property is set, CCSStudio will connect all the master devices on Startup, and then the slave devices.



### Using BYPASS Devices

CCS - L8

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

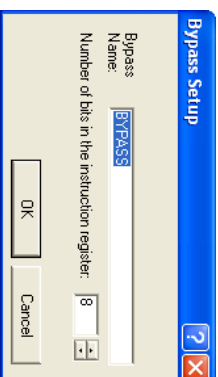
26/09/2007



In multiprocessor systems, you may sometimes wish to bypass certain devices that are a part of the JTAG scanpath from being targeted by the Code Composer Studio™ debugger. For example, if a particular board has two DSP chips on it, and you wish to program on only one of the chips, you may (in some cases) have to set the other chip as a “dummy” BYPASS chip. The host will not contact this chip. Only

1. Select a supported board in the System Configuration pane. Bypass is displayed in the Available Factory Boards pane.
2. Open the Bypass Setup dialog box by either double-clicking on the Bypass icon in the Available Factory Boards pane, or dragging and dropping Bypass from the Available Factory Boards pane onto the board in the System Configuration pane.
3. Enter the Bypass Name. You can either accept the default Bypass Name or specify a meaningful name.
4. Enter the number of bits in the device's JTAG instruction register. Consult your board documentation for this information.

**Note:** In DSP devices, this refers to the number of bits in the instruction register for your target device.



5. Move Bypass to its correct location in your system configuration. The order of devices in your system configuration should be the same as the order of devices on the JTAG scan path. Initially, Bypass will appear at the end of your configuration. Click and drag the Bypass icon to its correct location in the configuration.

**Note:** If you wish for a processor to be treated as a Bypass, right-click on the processor and choose Treat As Bypass from the context menu and choose the number of bits.

In the next lesson, you will learn a few tips for configuring target devices.



### Tips on Configuring Target Devices

CCS - L8

#### Startup GEL Files

After selecting a board and processor, you can specify a startup GEL File. Click on a supported processor and choose Properties from the context menu, then browse for the new GEL file.

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007

#### Getting Started With the Code Composer Studio Tutorial

Page 130 sur 548

This GEL file will then perform any initialization required on the target board. This may include, but is not limited to, resetting the DSP, initializing the External Memory Interface (EMIF), and resetting breakpoints. For more information about GEL, see the CCS online help.

#### Using GEL with DSP Reset

Create and load a GEL function called OnReset(int nErrorCode). This function is called whenever DSP Reset is chosen from the Debug menu. The nErrorCode flag indicates the status of the callback, where 0 indicates success.

**Note:** The GEL DSP Reset function was formerly called GEL\_ONDSPReset. However, the function has been changed to OnReset.

This concludes the Configuring Target Devices lesson.



### Application Code Tuning Introduction

Tune - Intro

One of the largest areas of concern in a DSP developer's development flow is the phase of tuning and optimization. DSP developers expect to achieve a very efficient solution, and often spend large amounts of time improving the efficiency of an application. They work in this phase as long as is necessary to achieve their efficiency goals, these being defined uniquely by each customer and their application. Efficiency may mean very different things to different customers. Common areas of concern include: cycle count performance and code size. Of course, optimizing for one factor may negatively or positively impact the other one. And furthermore, these two factors may impact a third factor, cost. Therefore, the Code Composer Studio™ IDE provides a set of tuning tools to help developers to quickly reach their optimization goals. The tuning tool suite is a cohesive set of tuning tools, each a strategic part of the DSP tuning phase, logically formed to attack most of the key areas in which DSP developers need high efficiency.

The tutorials demonstrate how you can optimize, or tune, some sample projects, by profiling your application's execution and using the various new tuning tools. The Tuning Dashboard collects these tools in a single location and provides advice on how to use the tools to increase application efficiency. You will quickly learn the functions and windows of the Dashboard, including the Profile Setup, the Goals Window, the Advice Window, and the Profile Viewer. You will also review how to profile your program effectively, and learn how to use tools such as CodeSizeTune, Compiler Consultant, and CacheTune.

Target Configuration: These profiling lessons were written for use with a simulator, and will not function properly with a DSK hardware configuration. For more information on hardware profiling limitations in Code Composer Studio 3.1, please see the online help .

This tutorial module contains the following lessons on each of the Tuning Tools:

[Tuning Dashboard](#)

[CodeSizeTune](#)

[Compiler Consultant](#)

[CacheTune](#)

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007

▶ The forward arrow will take you to the next page in this tutorial.

## Tuning Dashboard Introduction

### DASH - Intro

The Tuning Dashboard helps developers to apply the tuning tools as part of an organized optimization method.

Before using this tutorial module, you should have done the following:

- Installed Code Composer Studio™ IDE

**Target Configuration:** To run these tutorials, you must set up the Code Composer Studio™ IDE to run on the C6416 device cycle accurate simulator, little endian. The Optimization dashboard tool will also function on all 6x and 55x simulators. For 55x simulators, replace the sim64xx target references with sim55xx; the result numbers will also vary. For more information on the Setup utility, access the [Configuring Target Devices](#) tutorial or the online help.

This tutorial module contains the following lessons:

- Profile Setup
- Goals Window
- Advice Window
- Profile Viewer
- Tune an Application

▶ The forward arrow will take you to the next page in this lesson.

## Profile Setup Overview

### DASH - L1

#### Learning Objectives:

file:///C:/Documents and Settings/Pierre-Armand/Local Settings/Temp/~hhBA53.htm

26/09/2007

## Getting Started With the Code Composer Studio Tutorial

Page 132 sur 548

- Launch Profile Setup
- Identify common tuning activities
- Specify profiling collection options for functions, loops, and ranges using the Ranges tab
- Set exit, halt, and resume points in the Control tab
- Customize profile data collection using the Custom tab

#### Example used in this lesson: Modem

Target Configuration: The modem demonstration application is located in the [C:\CCStudio\\_v3.10\tutorial\target\modem](#) directory. To run this tutorial, you must setup the Code Composer Studio™ IDE to run on a 6416 device cycle accurate simulator, little endian. For more information about Setup, run Setup using the Setup CCS icon, and use the online help.

#### Application Objective:

This tutorial introduces the basic features of the Tuning Dashboard and takes you through a step-by-step approach for navigating through the profile setup process.

This lesson consists of the following steps. To go directly to a step, click on the link below:

- [Launching Profile Setup](#)
- [Activities Tab](#)
- [Ranges Tab](#)
- [Control Tab](#)
- [Custom Tab](#)

▶ The forward arrow will take you to the next page in this lesson.

## Launching Profile Setup

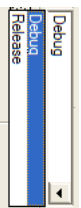
### DASH - L1

1. If you have not already done so, from the Windows Start menu, choose Programs → Texas Instruments → Code Composer Studio 3.1 → Code Composer Studio 3.1. (Or, double-click the Code Composer Studio icon on your desktop.)


file:///C:/Documents and Settings/Pierre-Armand/Local Settings/Temp/~hhBA53.htm

26/09/2007

- From the Project menu, choose Open, and browse to: `C:\CCStudio_v3.10\tutorial\sim64xx\modem`.
- Select `modem.pjt` and click Open. This opens an example tuning project.
- Under the Project menu, choose Build Options...
- In the Build Options dialog, click the Compiler tab.
- Verify that the Opt Level is set to None.
- Verify that the Generate Debug Info option has Full Symbolic Debug (-g) selected from the drop-down list, and click OK.



- From the Project menu, choose Rebuild All. An output file (.out) must be generated and loaded before beginning the tuning process.
- From the File menu, choose Load Program.
- Browse to the location of the output file, which should be: `C:\CCStudio_v3.10\tutorial\sim64xx\modem\Debug`.
- Select `modem.out` and click Open. When the example program has been loaded, the Disassembly window will open in the editor.

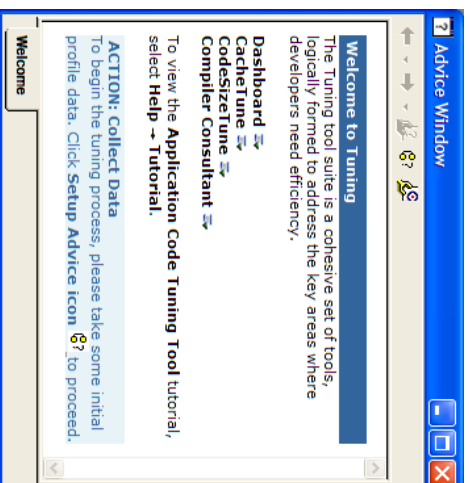
- Activate tuning mode by clicking on the tuning fork icon  or by clicking the menu item View->Layout->Tuning Layout. Switching to Tuning mode reorganizes the workspace to provide a good view of both the Project and the Advice windows, which helps to guide the tuning process. You have the option to float this window in your main workspace by right-clicking on the window and choosing 'Float' in main window' from the context menu.


file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007

Getting Started With the Code Composer Studio Tutorial

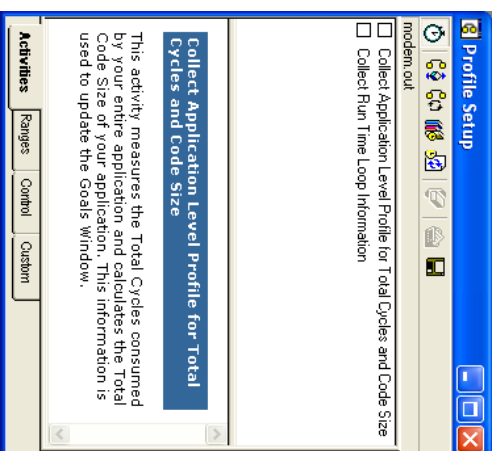
Page 134 sur 548




- In the Advice window, click on the View Tuning Setup Advice in Setup tab icon  and click on the link for the Profile Setup tool. This tool can also be opened using the Profile->Setup menu item. The Profile Setup opens on the right side of the screen and allows the setting of several options to customize the profiling of the application. You have the option to float this window in your main workspace by right-clicking on the window and choosing "Float in main window" from the context menu.

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007



15. Check the icons at the top of the Profile Setup tool to determine if profiling is enabled. The Enable/Disable Profiling icon  must be toggled on to enable profiling. Profiling may not be enabled if the program has started running or if it was explicitly disabled. Data will not be collected if profiling is not enabled.



### Activities Tab

DASH - L1

Use the Activities tab to select common tuning activities.

1. Click on the Activities tab of the Profile Setup window to select the profiling activities for collection. When you highlight or select an activity, that activity's description is displayed in the lower half of the Profile Setup window. If you cannot see the activity description, resize the upper portion of the window.
2. Click on the box beside Collect Application Level Profile for Total Cycles and Code Size to collect this information. This activity measures the total cycles consumed by the entire

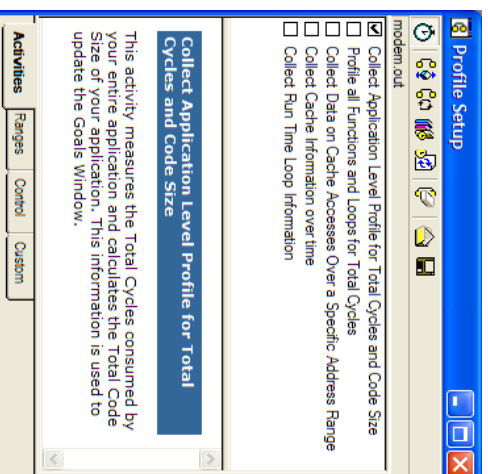
file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007

### Getting Started With the Code Composer Studio Tutorial

Page 136 sur 548

application and calculates the total code size of the application.



### Ranges Tab

DASH - L1

The Range tab specifies functions, loops, and ranges to be profiled.

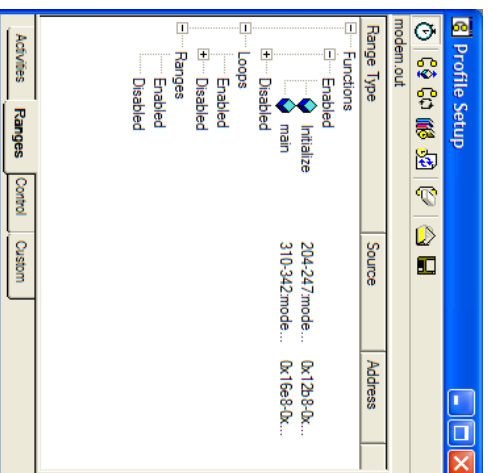
1. The Project View in the upper left corner of the Tuning Layout displays the files available in the modern project. Click on the project name to open the view.
2. Double-click on modern\c under the source branch to open the source file in the editor. This file contains most of the source code of interest, so it is helpful to have it open.
3. Click the Ranges tab of the Profile Setup window. This tab specifies and displays the lines of code for which data is collected. Profile data can be collected for functions, loops, and


file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007

ranges of source code.

- Highlight the Initialize function in modemtx.c. Add this function to the list of ranges by right-clicking on it, and selecting Profile→Functions In Range from the context menu. The Initialize function should appear in the Functions branch of the Ranges tab. The main function may also appear.



- Quickly profile all functions in the program, click the Enable/Disable All Functions icon  at the top of the Ranges tab to toggle it on. The program's functions should appear in the Functions/Enabled branch of the Ranges tab.
- Add a loop by right-clicking on the Profile Setup window, and choosing Create Profile Item. The source file path should appear in the Add Profile Item window because it is already open in the editor.
- In the Type drop-down box, select All Loops to profile each loop in the selected function.
- To specify the Profiling Range, select Symbol Name, and enter ShapingFilter as the Function Name to select the entire ShapingFilter function.
- Click OK. The loops should appear in the Loops branch of the Ranges tab.
- Highlight the "for" loop in the AddNoiseSignal function of modemtx.c. Add this range to the Profile Setup by dragging it into the Ranges tab and dropping it on the Ranges branch of

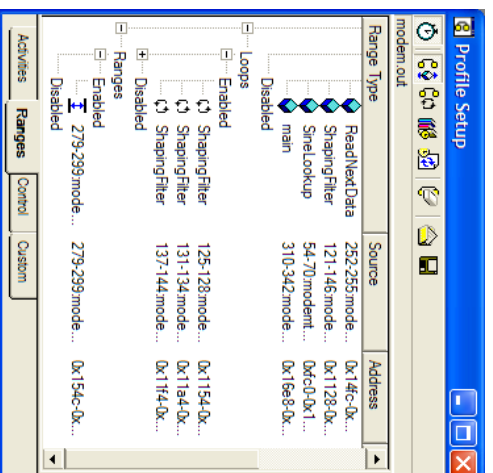
file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007

Getting Started With the Code Composer Studio Tutorial

Page 138 sur 548

the tree.



- To remove ranges from the Ranges tree, hold down the Control key and click on the ranges. Press the Delete key to remove the selected ranges. To remove a list of ranges, click on the first range, hold down the Shift key and click on the last range before pressing Delete.
- To disable all the functions and loops, toggle the Enable/Disable All Functions and the Enable/Disable All Loops buttons off. You can also disable one or more functions or loops by selecting a function or loop and either pressing the space key on your keyboard, or right-clicking and choosing the context menu item Enable/Disable Selected Item(s).
- You can also move disabled functions or loops to enabled by selecting them and pressing the space key on the keyboard, or right-clicking and selecting the context menu item Enable/Disable Selected Item(s).
- To prepare for the Goals Window lesson, remove or disable all entries in the tree with the exception of the Initialize function.



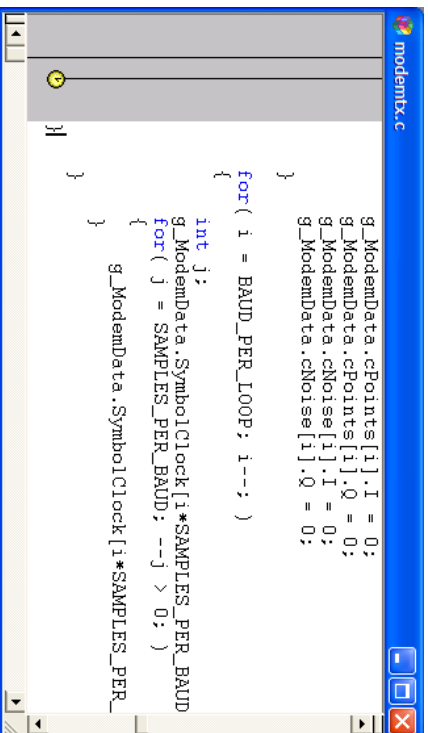
**Control Tab**

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007

Use the Control tab to set exit, halt, and resume points. Exit points are used to stop collecting profile data for the application. Typically, this stopping point is when the application stops executing. However, an exit point can be set at any place in the code. It is necessary to add an exit point to stop data collection for an application that runs in an infinite loop. Halt points are used to temporarily stop collecting profile data for the application and can be set anywhere in the code. Resume points are used to continue the data collection process after a halt point has been reached. This ability to have data collection "skip" over certain code can be useful for excluding such areas as code initialization or test harnesses.

1. Click on the Control tab of the Profile Setup window. This tab provides a way to set exit, halt, and resume points in the application.
2. Move the cursor to the last line (the closing brace) of the Initialize function of `modemtx.c` to determine its line number (247). Your line number may vary.



```

g_ModemData.cPoints[i].I = 0;
g_ModemData.cPoints[i].Q = 0;
g_ModemData.cNoise[i].I = 0;
g_ModemData.cNoise[i].Q = 0;
}
for ( i = BAUD_PER_LOOP; i--> )
{
    int j;
    g_ModemData.SymbolClock[i*SAMPLES_PER_BAUD
for ( j = SAMPLES_PER_BAUD; --j > 0; )
    {
        g_ModemData.SymbolClock[i*SAMPLES_PER_
    }
}
}

```

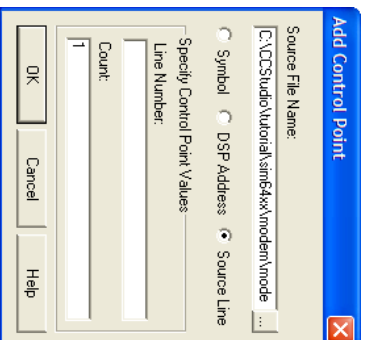
3. To add an exit point to the application, right-click on the Control tab and select Create Exit Point from the context menu. The source file path should appear in the Add Control Point window because it is already open in the editor.
4. Click on the Source Line radio button to select it.

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007

Getting Started With the Code Composer Studio Tutorial

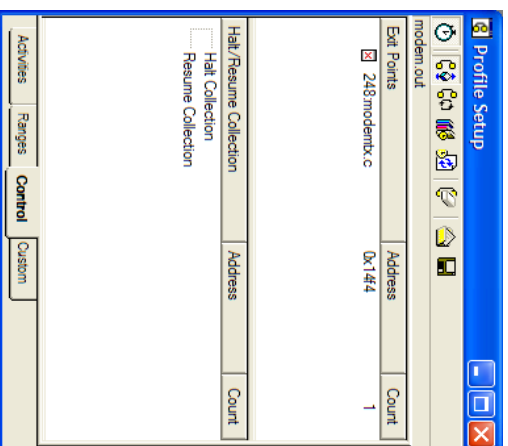
Page 140 sur 548



5. In the Line Number box, enter the line number that corresponds to the last line of code in the Initialize function (line 247).
6. In the Count box, enter 1 as the number of code profiling iterations. This instructs data to be collected for a maximum of one iteration. When the line corresponding to the exit point has been run once, data collection will stop.
7. Click OK to add the exit point. The new exit point should appear in the list of exit points in the upper section of the Control tab.

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007



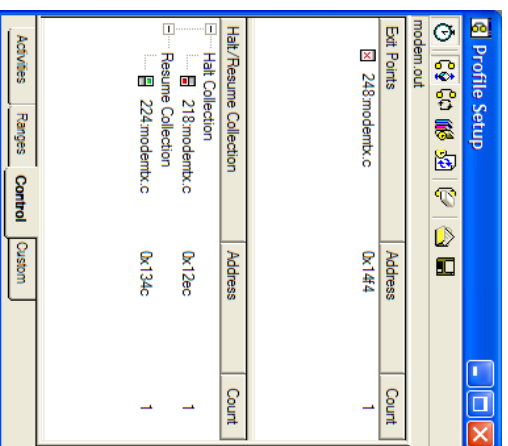
8. Select the line that begins the first "for" loop in the Initialize function.
9. To add this line as a halt point in the application, drag the selected line into the Halt Collection pane in the Control tab. Data collection will now stop when the first "for" loop is reached.
10. Resume points will resume data collection if a halt point has been reached. Add a resume point at the beginning of the second for-loop in the same fashion as the halt point in the previous two steps.

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007

Getting Started With the Code Composer Studio Tutorial

Page 142 sur 548



11. To remove a control point, select the point and click Delete.



**Custom Tab**

**DASH - L1**

Use the Custom tab to customize profile data collection.

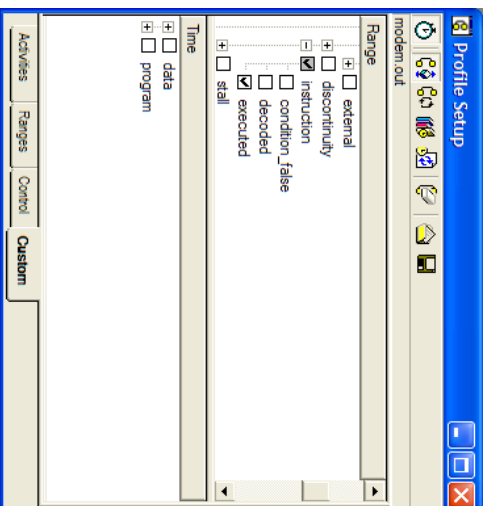
1. Click on the Custom tab of the Profile Setup window to manually select the data to be collected. This tab allows for the selection of individual events to be profiled in the application.
2. Range events (found at the top of the tab) are counted as they occur over the ranges selected in the Ranges tab. Select the following events in the Range tree on the Custom tab for profiling:
  - CPU→access→summary

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007

- XTIY →wcpuxrtw→e5cpu8

You may need to resize the profile setup window to view all the events. Selecting these events will count the number of CPU accesses and instruction executions that occur in the Initialize function. Please note that these events may change for different ISA configurations.



- Time events (found at the bottom of the custom tab) are logged and counted while the application is running. If a selected event occurs, the time it occurred is recorded for later review. Time events are used only for the CacheTune tool, which provides a graphical visualization of cache accesses over time. You may need to resize the profile setup window to view all the events. Selecting the following events will record the occurrences of CPU data reads and program cache misses in L2. Select these events in the Time tree for profiling on a C64x simulator:
    - data →XTIY →access →data →read
    - program →XTIY →instruction →executed
    - program →A2 →xxqjre →miss →summary
- For a C620x (not C6211) or a C6701 simulator, choose the following events to record the occurrences of CPU data reads and CPU instruction executions:
- data →XTIY →access →data →read

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

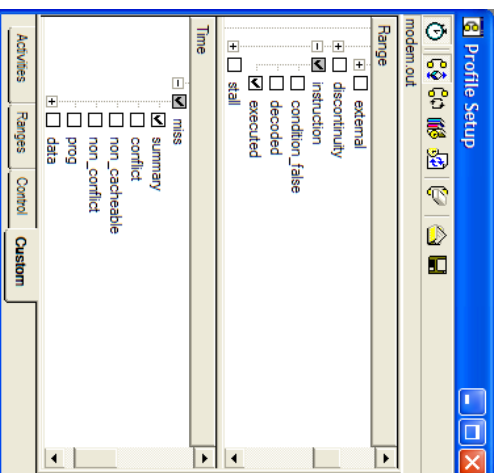
26/09/2007

Getting Started With the Code Composer Studio Tutorial

Page 144 sur 548

- program →CPU →instruction →executed

Please note that these events will change for different ISA configurations.



- Be sure to un-check the Activities boxes, delete or disable any functions or ranges other than the Initialize function, remove all control points, and un-check the Custom event boxes before moving to the next lesson. You can also click on the Clear Profile Configuration  button in the toolbar to disable or delete this configuration.
- You can optionally right-click on the Profile Setup window and click Hide to close the window.

The next lesson in the Tuning Dashboard tutorial reviews the [Goals Window](#) tool.



[Goals Window Overview](#)



**Learning Objectives:**

- Launch the Goals window
- Set goals based on the needs of the application
- Track tuning progress through a goals log
- Interpret data in the Goals window

**Example used in this lesson: Modem**

**Target Configuration:** The modem demonstration application is located in the C:\CCStudio\_v3.10\tutorial\sim64xx\modem directory. To run this tutorial, you must setup the Code Composer Studio™ IDE to run on a 6416 device cycle accurate simulator, little endian. For more information about Setup, run Setup using the Setup CCS icon, and use the online help.

**Application Objective:**

This tutorial introduces the Goals window in the Tuning Dashboard and demonstrates a step-by-step process to setup and track performance goals.

This lesson consists of the following steps. To go directly to a step, click on the link below:

[Launching the Goals Window](#)

[Setting Application Goals](#)

[Tracking Tuning Progress](#)

[Interpreting Goals Data](#)

**Launching the Goals Window**

DASH - L2

1. If you have not already done so, from the Windows Start menu, choose Programs→Texas Instruments→Code Composer Studio 3.1→Code Composer Studio 3.1. (Or, double-click the Code Composer Studio icon on your desktop.)
2. From the Project menu, choose Open, and browse to: C:\CCStudio\_v3.10\tutorial\sim64xx\modem.
3. Select modem.pjt and click Open. This opens an example tuning project.

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007

**Getting Started With the Code Composer Studio Tutorial**

Page 146 sur 548

4. From the File menu, choose Open to browse to the modemtx.c source file and open it in the editor.
5. Find the line before the first for-loop (line 215) in the Initialize function. Your line number may vary.



```

*****
void Initialize(void)
{
    int i;
    #if defined(C54X)
        asm("RSBX OVM");
    #endif
    g_ModemData.carrierFreq = 15; /* increment th
    g_ModemData.phase = 0;
    g_ModemData.samplesPerBand = SAMPLES_PER_BAUD;
    g_ModemData.noiseLevel = 0;
    /* zero delay lines for shaping filter */
    for( i = SIZE SHAPING_FILTER; i--> )

```

6. Insert the following code before the first for-loop (line 215) in the Initialize function. Your line number may vary:

```

For (i = 0; i < 100; i++)
{
    g_ModemData.noiseLevel = 0;
}

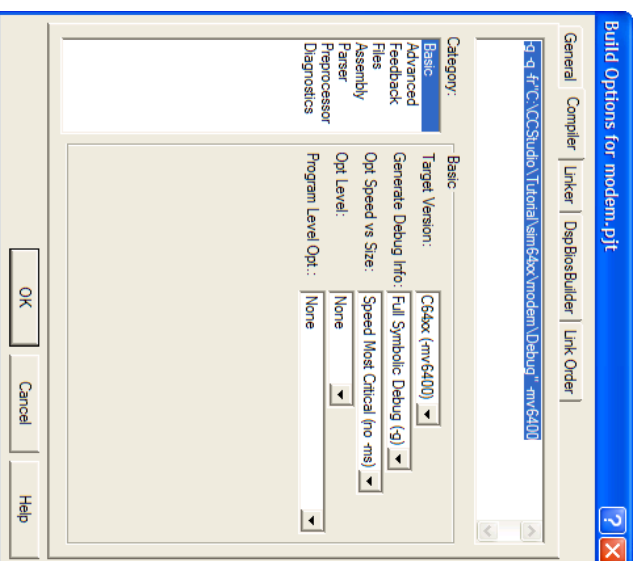
```


This deliberate introduction of inefficient code allows us to see how the data in the Goals window reflects the code itself.

7. From the File menu, choose Save to save the file.
8. Select Project→Build Options. On the Compiler Tab, change the optimization level (Opt Level) to None. Part of the compiler's optimization capability is to recognize code that performs no function and to eliminate it. Since the inefficient code has been introduced to demonstrate the effect on the goals window, the optimizer must be turned off to be prevented from removing it. Click OK to save the setting.

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007



9. From the Project menu, choose Rebuild All. An output file (.out) must be generated and loaded before beginning the tuning process.
10. From the File menu, choose Load Program.
11. Browse to the location of the output file, which should be: `C:\CCStudio_v3.10\tutorial\sim64xx\modern\Debug`.
12. Select `modern.out` and click Open. When the example program has been loaded, the Disassembly window will open in the editor.
13. If not already active, activate tuning mode by clicking on the tuning fork icon  or by clicking the menu item `View-->Layout-->Tuning Layout`. Switching to Tuning mode reorganizes the workspace to provide a good view of both the Project and the Advice windows, which helps to guide the tuning process. If the Advice Window is not opened, choose

<file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm>

26/09/2007

Getting Started With the Code Composer Studio Tutorial

Page 148 sur 548

Tuning-->Advice Window.


14. In the Advice Window, click on the View General Tuning Advice icon  and click on the link to launch the Goals Window. This window can also be opened using the `Profile-->Tuning-->Goals` menu item.



### Setting Application Goals

**DASH - L2**

You will now set goals based on the needs of the application

1. Use the Profile Setup tool to select the profiling activity. Collect Application Level Profile for Total Cycles and Code Size. This activity will collect data for total cycles and code size. Please see the [Activities Tab](#) lesson in Profile Setup for more information.
2. Use the Profile Setup tool to add an exit point at the last line of code (line 264) of the ReadConstellation function belonging to the Initialize function. Your line number may vary. When the exit point has been reached, data collection will stop. Please see the [Control Tab](#) lesson in Profile Setup for more information.
3. Check the icons at the top of the Profile Setup tool to determine if profiling is enabled. The Enable/Disable Profiling icon  must be toggled on to enable profiling. Profiling may not be enabled if the program has started running or if it was explicitly disabled. Data will not be collected if profiling is not enabled.
4. Select the Debug--Run menu item or press F5 to run the application. This will generate initial data values in the Current column of the Goals window. These values represent the Code Size and Cycle Total. The Code Size is the total number of bytes required to store the program instructions. The Cycle Total is the total number of cycles required to execute the instructions in the application, including both memory accesses and input/output operations when applicable. In this example, Code Size is the number of bytes used to store the Initialize function or whole application, and Cycle Total is the number of cycles used to execute the function.
5. When the values in the Current column of the Goals Window have been updated, select Debug--Halt or press Shift-F5 to Halt the application. The initial set of data for the Initialize function has now been collected. Your values may vary.

<file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm>

26/09/2007

Code Size	Goal	Current	Previous	Delta
Cycle Total		17497	-	-

- The Goal column of the Goals Window stores the desired Code Size and Cycle Total values. These goals should be set depending on the needs of the application, which may, in turn, be dependent upon limitations imposed by hardware, application requirements, or other software. For example, if a DSP has 40 KB of RAM available for an application, a Code Size goal of 20,000 bytes may be appropriate. For the purposes of this example, enter a value in the Goal column for Code Size, roughly 100 bytes less than the Current Code Size.
- Code may also need to be limited to executing in a certain number of cycles. For instance, hardware may require a sort algorithm to complete in 0.05 seconds. In the case of a 600MHz processor, the Cycle Total goal for the range of code that performs the sort must be 30,000,000. For the purposes of this example, enter a value in the Goal column for Cycle Total that is roughly 100 cycles lower than the Current Cycle Total.


Code Size	Goal	Current	Previous	Delta
Cycle Total		17397	17497	-



### Tracking Tuning Progress

DASH - L2

Tuning progress can be tracked on an application level using the Goals Log.

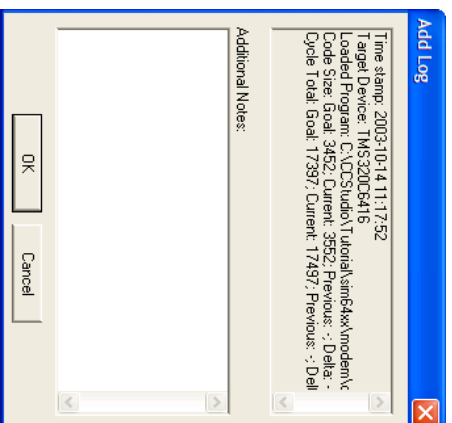
- To capture the current state of the Goals window by adding an entry to the log, click on the Add Log Entry icon  in the Goals window.
- The Add Log window will appear.

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm


26/09/2007

Getting Started With the Code Composer Studio Tutorial

Page 150 sur 548

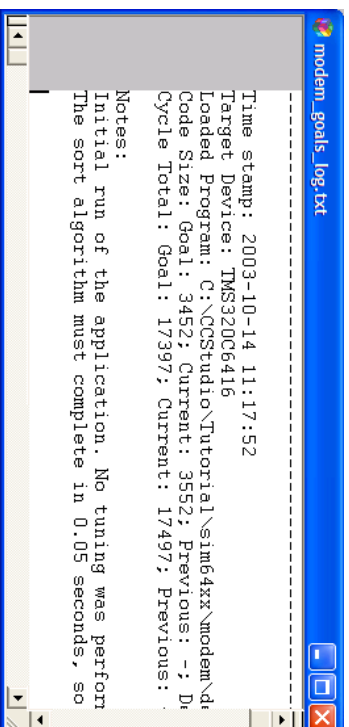


The upper section of the window shows the tuning information to be included in the log entry. This includes the time of the log entry, target device, program, and the data values from the Goals Window.

- The lower section of the Add Log window provides the opportunity to record comments in the log entry. Type a comment into the Additional Notes section of the Add Log dialog box. A helpful comment at this stage in the tuning process might be: "Initial run of the application. No tuning was performed at this point." It may also be useful to include reasons for the tuning goals. For example, "The sort algorithm must complete in 0.05 seconds, so on a 600 MHz processor the Cycle Total must be less than 30,000,000."
- Click OK to add the entry to the tuning log.
- The tuning log can be examined at any time. Click the View Log icon  in the Goals window. The tuning log will be displayed in the editor. The log is stored in a text file called <project>\_goals\_log.txt (in this case, modem\_goals\_log.txt). All log entries for this project will be maintained in this log. The tuning log can be edited and saved in the same manner as other files in the Code Composer Studio™ IDE.

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007



- Close the log file by clicking File→Close. If changes have been made to the log file while it was open in the editor, a prompt window will ask whether to save or discard the changes. Click Yes to save the changes or No to discard them.



### Interpreting Goals Data

DASH - L2

At this point in the tutorial, the goals for Code Size and Cycle Total are less than the current values for the application. When the application does not meet the goals that have been set, the collected data appears red and italicized. This should be the case with the values in the Current column.

	Goal	Current	Previous	Delta
Code Size	3452	<i>3552</i>	-	-
Cycle Total	17397	<i>17497</i>	-	-

- Click on the Debug→Reset CPU menu item. Then click on File→Reload Program. Resetting and reloading is important because it informs the Goals window that another execution of the application is about to begin. If the debugging process is not restarted, any new data will be added to the current data in the Goals Window.

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007

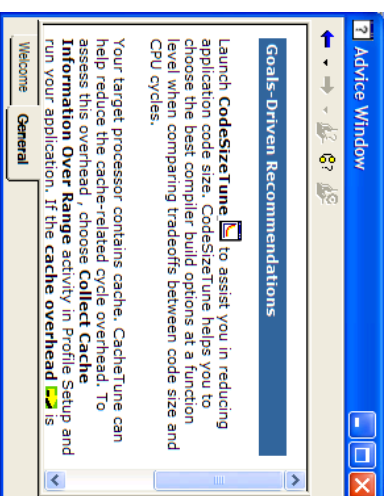
### Getting Started With the Code Composer Studio Tutorial


Page 152 sur 548

- From the Debug menu, choose Run. The new data is displayed in the Goals Window as it is collected. The data from the last execution of the program moves to the Previous column and the difference between values is displayed in the Delta column.
- From the Debug menu, choose Halt. As the goals have not yet been reached, the current data values are still shown in red, italicized text. Your numbers may vary.

	Goal	Current	Previous	Delta
Code Size	3452	<i>3552</i>	<i>3552</i>	0
Cycle Total	17397	<i>17497</i>	<i>17497</i>	0

The Advice window will now display one or more tips for meeting the goals. For example, if the size of the code must be reduced, the Advice window may suggest using the CodeSizeTune tool and provide a link to it. If the number of cycles must be reduced, the Advice window may suggest using the Compiler Consultant and provide a link to it. Suggestions in the Advice window can be valuable aids for tuning an application.



- Create a log entry by clicking on the Add Log icon  and, after typing in a comment, clicking OK.
- Remove the first "for" loop in the Initialize function, added at the beginning of this lesson. When this code is removed, the Initialize function should meet the cycle goals that have been set.
- From the File menu, choose Save to save this file.

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007

7. Click on the Project→Rebuild All menu item to rebuild the application. The new executable will contain the changes made in the previous step.
8. Reload the executable file by clicking File→Reload Program. This will ensure that the latest version of the executable has been loaded.
9. From the Debug menu, choose Run to run the modified version of the application. In the Goals window, the values from the Current column should appear in the Previous column and the Current column should be updated with new data collected from the application.
10. From the Debug menu, choose Halt. If the application has met the goals set earlier in the lesson, the values for Code Size and Cycle Total will be green and appear in parentheses.



Goal	Current	Previous	Delta
Code Size	3452	3488	-64
Cycle Total	17397	(15087)	17497

If the goals have not been met, the values will continue to be red.

11. Create a log entry to record the results of this tuning period, by clicking on the Add Log icon  and, after typing in a comment, clicking OK.
12. Right-click on the Goals window and click Hide to close the window.
13. Since the optimization level was set to None at the beginning of this lesson, it must now be reset. Select Project→Build Options. On the Compiler tab, change the optimization level (Opt Level) to File (-O3). Click OK to save the setting.

The next lesson in the Tuning Dashboard tutorial reviews the [Advice Window](#) tool.



### Advice Window Overview

DASH - L3

#### Learning Objectives:

- Launch the Advice Window.
- Navigate in the Advice Window
- Identify the types of available advice

file:///C:/Documents and Settings/Pierre-Armand/Local Settings/Temp/~hhBA53.htm

26/09/2007

### Getting Started With the Code Composer Studio Tutorial

Page 154 sur 548

- Get advice on specific tuning tools

#### Example used in this lesson: Modern

Target Configuration: The modern demonstration application is located in the C:\CCStudio\_V3.10\tutorial\sim64xxx\modern directory. To run this tutorial, you must setup the Code Composer Studio™ IDE to run on a 6416 device cycle accurate simulator, little endian. For more information about Setup, run Setup using the Setup CCS icon, and use the online help.

#### Application Objective:

This lesson introduces the Advice window in the Optimization Dashboard and takes you through a step-by-step process for navigating tuning.

This lesson consists of the following steps. To go directly to a step, click on the link below:

- [Launching the Advice Window](#)
- [Navigating in the Advice Window](#)
- [Advice Types Introduction](#)
- [Tuning Tool-Specific Advice](#)



### Launching the Advice Window


DASH - L3

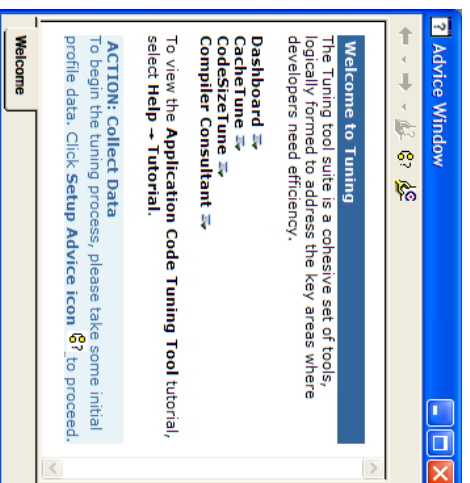
1. If you have not already done so, from the Windows Start menu, choose Programs→Texas Instruments→Code Composer Studio 3.1→Code Composer Studio 3.1. (Or, double-click the Code Composer Studio icon on your desktop.)
2. From the Project menu, choose Open, and browse to: C:\CCStudio\_V3.10\tutorial\target\modern.
3. Select modern.pjf and click Open. This opens an example tuning project.
4. From the Project menu, choose Rebuild All. An output file (.out) must be generated and loaded before beginning the tuning process.
5. From the File menu, choose Load Program.
6. Browse to the location of the output file, which should be: C:\CCStudio\_V3.10\tutorial\target\modern\Debug.

file:///C:/Documents and Settings/Pierre-Armand/Local Settings/Temp/~hhBA53.htm

26/09/2007

7. Select **modem.out** and click **Open**. When the example program has been loaded, the **D**isassembly window will open in the editor.

8. If not already active, activate tuning mode by clicking on the tuning fork icon  or by clicking the menu item **View->Layout->Tuning Layout**. Switching to tuning mode reorganizes the workspace to provide a good view of both the **Project** and the **Advice** windows, which helps to guide the tuning process. The **Welcome** tab in the **Advice** Window will open automatically when **Tuning Layout** is selected. You can also launch the **Advice** Window by clicking on **Profile->Tuning->Advice**.



### Navigating in the Advice Window

DASH - L3


1. Click on the **Welcome** tab of the **Advice** window. If not already selected, the **Welcome** page describes the tools that are available for a particular target processor. Such tools may include **CacheTune**, **CodeSizeTune**, and the **Compiler Consultant**.
2. As is the case with many popular web browsers, information in the **Advice** window contains underlined links that, when clicked, navigate to a new location in the information.

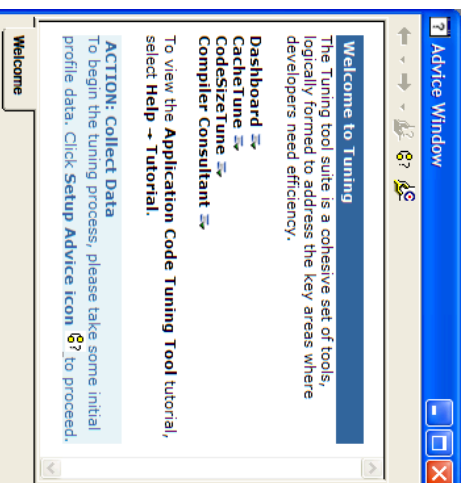
file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm


26/09/2007


### Getting Started With the Code Composer Studio Tutorial

Page 156 sur 548

3. There are also links (  ) that open a drop-down section in the window with more information.
4. The **Advice** window highlights a step in the tuning process by surrounding a clear description of the action in a blue shadow box, highlighting the keyword **Action**. Scroll down to the bottom of the **Welcome** page. The **Advice** window indicates that the next step is to collect data for the application. It also provides an icon link to the **Setup Advice** page.



5. Click on this link to open the **Setup Advice** page. The **Setup Advice** page can also be opened by clicking on the **Setup Advice** icon  at the top of the **Advice** window.
6. Scroll to the bottom of the **Profile Setup** advice, and click on the link **Back to Top**. This will display the top of the **Setup Advice** page again.

7. When active, the **backward-pointing arrow**  navigates to past locations in reverse-chronological order. **Tuning** tool-specific advice often has several pages, so the arrows allow you to easily navigate the sections. The arrows are not active for the introductory pages. Try opening other advice pages to see the arrows activate.

**Tip:** Click on the small down-arrow beside the icon to choose from a list of locations.

8. When active, the **forward-pointing arrow**  navigates to locations that were displayed prior to clicking the **back** arrow. The arrows are not active for the introductory pages. Try opening other advice pages to see the arrows activate.

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007

Tip: Click on the small down-arrow beside the icon to choose from a list of locations.

9. Return to the Welcome page by clicking on the Welcome tab at the bottom of the Advice window.



## Advice Types Introduction

DASH - L3

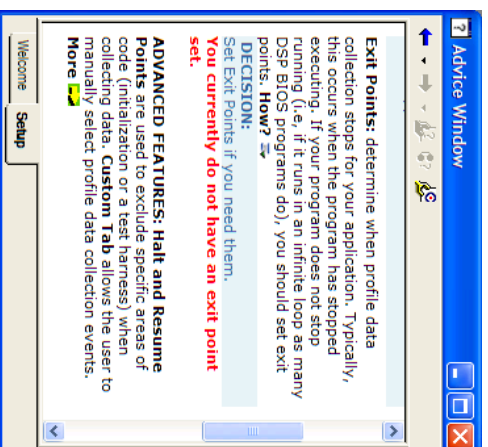
1. Activate the Setup Advice page by clicking on the Setup tab in the Advice window. The Setup Advice page provides a guide through each step of the tuning setup process and offers information on each of the components found in profile setup.
2. Click on the first link on the Setup Advice page to launch the Profile Setup tool. The Profile Setup tool allows the configuration of the type and range of the data collected.
3. Select the profiling activity, Collect Application Level Profile for Total Cycles and Code Size, from the Activities tab. This activity will collect data for the tools used in the next steps of the lesson.
4. The Advice window can also diagnose problems with the tuning process. In the Setup Advice page, scroll down to the section describing exit points. This section will display a warning in red text indicating that an exit point has not yet been set.


file:///C:/Documents and Settings/Pierre-Armand/Local Settings/Temp/~hhBA53.htm

26/09/2007

Getting Started With the Code Composer Studio Tutorial

Page 158 sur 548



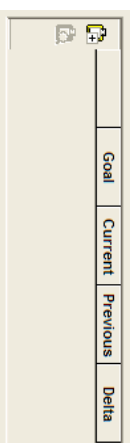
5. The Advice window will also provide instructions below the warning for adding an exit point. Follow the instructions to add an exit point at an arbitrary location in the code. Please see the [Control Tab](#) lesson in Profile Setup for more information. The warning will disappear once the exit point has been set.
6. Run the application by pressing F5.
7. Click on the View General Tuning Advice icon  at the top of the Advice window. This will open the Setting Goals section. The Setting Goals section describes the Goals window, a tool that tracks high level application goals and progress in meeting them.

file:///C:/Documents and Settings/Pierre-Armand/Local Settings/Temp/~hhBA53.htm

26/09/2007



8. Click on the  icon in the Setting Goals section to launch the Goals window. The Goals window will be displayed above the Advice window. If you did the Goals Window lesson before this one, the information from that lesson will still be displayed. You can also launch the Goals window by clicking on Profile→Tuning→Goals in the main menu.



9. The Advice window will display the Goals-Driven Recommendations section. This section suggests tools to use to meet tuning goals. For example, if one of the goals is a lower number of total cycles, the Advice window might recommend launching the Compiler Consultant to determine how to reduce the number of cycles.
10. Click on Debug→Halt to stop the debug process.



### Tuning Tool-Specific Advice

**DASH - L3**

1. Each tuning tool has a page in the Advice window that describes the tool, provides links to help, and gives useful steps to follow. Open the CacheTune tool by selecting the

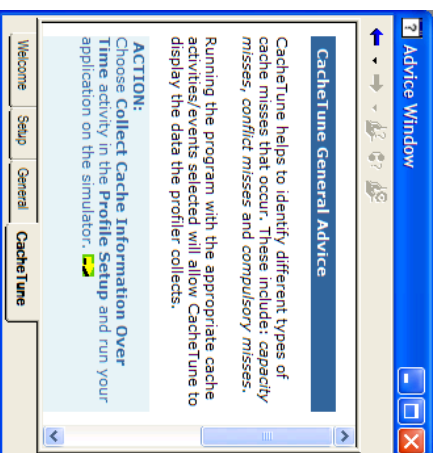
file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007

### Getting Started With the Code Composer Studio Tutorial

Page 160 sur 548

- Profile→Tuning→CacheTune menu item. The CacheTune tool provides a graphical visualization of cache accesses over time. The tool will open in the main editor window.
2. When the CacheTune tool has opened, the Advice window will open a new tab to display the General Advice for CacheTune page. The page describes the tool, and provides suggestions for getting the most out of the tool. All tuning tools have similar Advice pages that will be displayed when the tool is opened. Leave the Advice window open while tuning because it will display tool-specific advice during the entire process.



The next lesson in the Tuning Dashboard tutorial reviews the [Profile Viewer](#) tool. Be sure to un-check the Activities boxes and close any open windows before moving to the next lesson.



### Profile Viewer Overview

**DASH - L4**

#### **Learning Objectives:**

- Launch the profile viewer

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007



- Describe the components in the default data view
- Customize data views
- Sort data to find important information
- Save and restore data sets

**Example used in this lesson: Modem**

Target Configuration: The modem demonstration application is located in the `C:\CCStudio_v3.10\tutorial\sim64xx\modem` directory. To run this tutorial, you must setup the Code Composer Studio™ IDE to run on a 6416 device cycle accurate simulator, little endian. For more information about Setup, run Setup using the Setup CCS icon, and use the online help.

**Application Objective:**

This tutorial introduces the basic features of the Tuning Dashboard and takes you through a step-by-step approach for viewing and interpreting collected data displayed in the Profile Viewer.

This lesson consists of the following steps. To go directly to a step, click on the link below:

[Launching Profile Viewer](#)

[Default Data View Description](#)

[Customize Data Views](#)

[Sorting Data](#)

[Saving and Restoring Data Sets](#)



The forward arrow will take you to the next page in this lesson.

**Launching Profile Viewer****DASH - L4**

1. If you have not already done so, from the Windows Start menu, choose Programs→Texas Instruments→Code Composer Studio 3.1→Code Composer Studio 3.1. (Or, double-click the Code Composer Studio icon on your desktop.)
2. From the Project menu, choose Open, and browse to: `C:\CCStudio_v3.10\tutorial\sim64xx\modem`.


file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

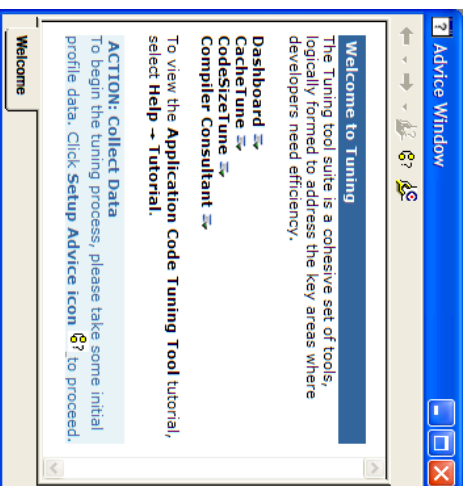
26/09/2007


**Getting Started With the Code Composer Studio Tutorial**

Page 162 sur 548

3. Select `modem.pjt` and click Open. This opens an example tuning project.
4. From the Project menu, choose Rebuild All. An output file (.out) must be generated and loaded before beginning the tuning process.
5. From the File menu, choose Load Program.
6. Browse to the location of the output file, which should be: `C:\CCStudio_v3.10\tutorial\sim64xx\modem\Debug`.
7. Select `modem.out` and click Open. When the example program has been loaded, the Disassembly window will open in the editor.



8. Activate tuning mode by clicking on the tuning fork icon  or by clicking the menu item View→Layout→Tuning Layout. Switching to Tuning mode reorganizes the Code Composer Studio workspace to provide a good view of both the Project and the Advice Window, which helps to guide the tuning process. You can float this window in your main workspace by right-clicking on the window and choosing Float in main window from the context menu.

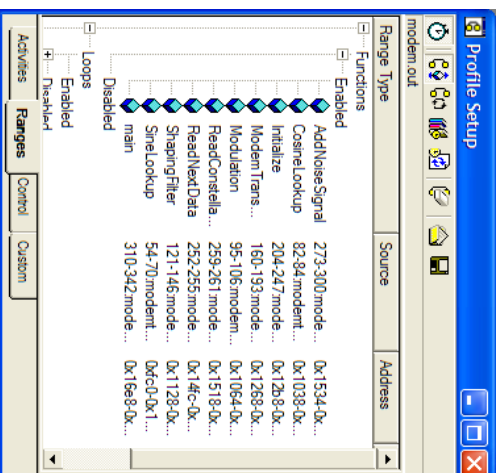


9. In the Advice window, click on the Setup Advice icon  and click on the link for the Profile Setup tool. This tool can also be opened using the Profile→Setup menu item. The Profile Setup opens on the right side of the screen and allows the setting of several options for data configuration. You can float this window in your main workspace by right-clicking on the window and choosing Float in main window from the context menu.

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007

10. Check the icons at the top of the Profile Setup tool to determine if profiling is enabled. The Enable/Disable Profiling icon  must be toggled on to enable profiling. Profiling may not be enabled if the program has started running or if it was explicitly disabled. Data will not be collected if profiling is not enabled.
11. In the Activities page of the Profile Setup tool, select the profiling activity Collect Application Level Profile for Total Cycles and Code Size. This activity will collect data for total cycles and code size. Please see the [Profile Setup](#) lesson for more details.
12. Click on the Ranges tab of the Profile Setup tool. Click on the Enable/Disable All Functions icon . Click on the plus (+) sign next to the Functions branch to see all of the functions in the application. If the functions are disabled, select them and press the space key to move them to enabled.



13. The Project View in the upper left corner of the Tuning Layout displays the files available in the modem project. Navigate to the Source folder and double-click on modemx.c to open the source file in the editor.
14. Add one or more ranges by highlighting a section of source code, dragging it to the Ranges tab, and dropping it on the Ranges branch. For instance, add the contents of the "for" loop in the AddNoiseSignal function.

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007

Getting Started With the Code Composer Studio Tutorial

Page 164 sur 548

```

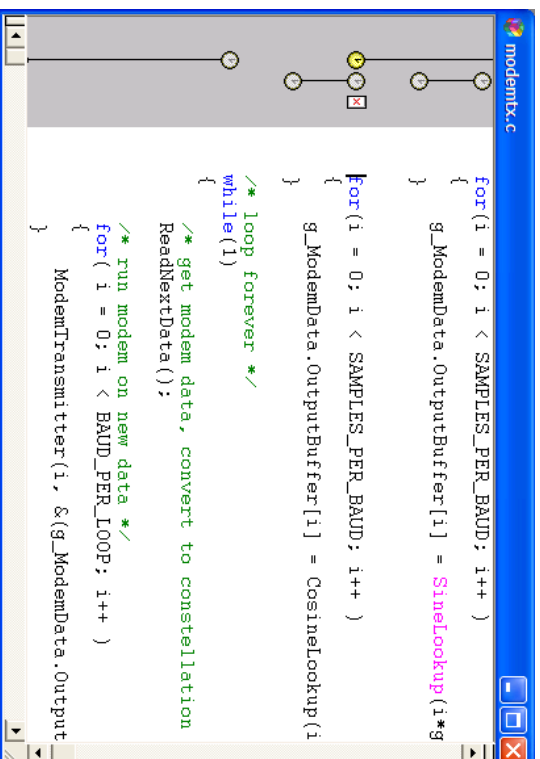
modemtx.c
{
    for( i = BAUD_PER_LOOP; i--; )
    {
        /* convert data to constellation points */
        g_ModemData.cPoints[i] = constellation[g_Mo
        /* Add noise to constellation points, only i
        {
            if (g_ModemData.noiseLevel != 0)
            {
                if (noiseVolume < 0)
                {
                    /* if volume is negative, shift to t
                    g_ModemData.cPoints[i].I += (g_Modem
                    g_ModemData.cPoints[i].Q += (g_Modem
                }
            }
            else
            {
                g_ModemData.cPoints[i].I += (g_Modem
                g_ModemData.cPoints[i].Q += (g_Modem
            }
        }
    }
}

```

15. You can delete ranges by clicking on the range and hitting Delete on your keyboard. You can disable functions and loops by clicking on the function or loop and hitting the space key on your keyboard.
16. Use the Control tab of the Profile Setup tool to add an exit point at the second "for" loop in the main function. The exit point will cause data collection to stop when it is reached. Please see the [Profile Setup](#) lesson for more details.

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007



```

modemTx.c
for(i = 0; i < SAMPLES_PER_BAUD; i++)
{
    g_ModemData.OutputBuffer[i] = SineLookup(i*g
}
}
For(i = 0; i < SAMPLES_PER_BAUD; i++ )
{
    g_ModemData.OutputBuffer[i] = CosineLookup(i
}
}
/* loop forever */
while(1)
{
    /* get modem data, convert to constellation
    ReadNextData();
}
/* run modem on new data */
for( i = 0; i < BAUD_PER_LOOP; i++ )
{
    ModemTransmitter(i, &(g_ModemData.Output
}

```

17. Click on the Custom tab of the Profile Setup tool and select the following tuning items:

- CPU->discontinuity->interrupt
- CPU->instruction->executed

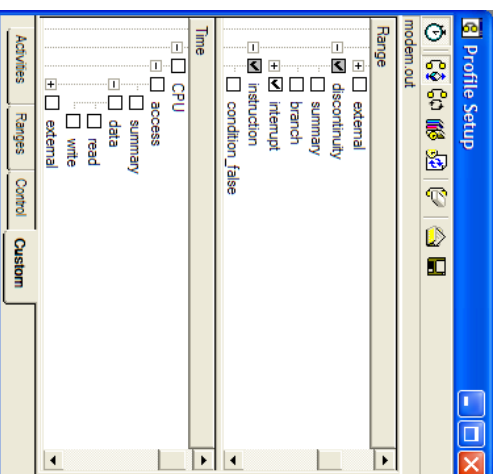
The cycle->Total item should already be selected.

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007

Getting Started With the Code Composer Studio Tutorial

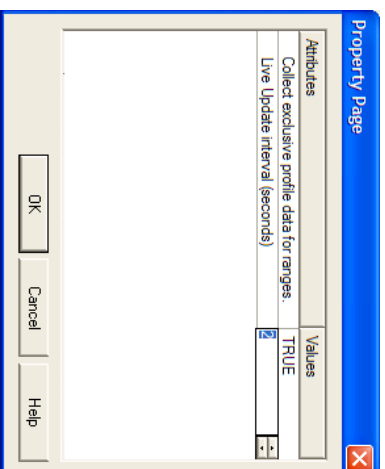
Page 166 sur 548



18. Right-click on each of the selected items and click on Property Page. Change the Live Update attribute to two seconds and click OK.

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

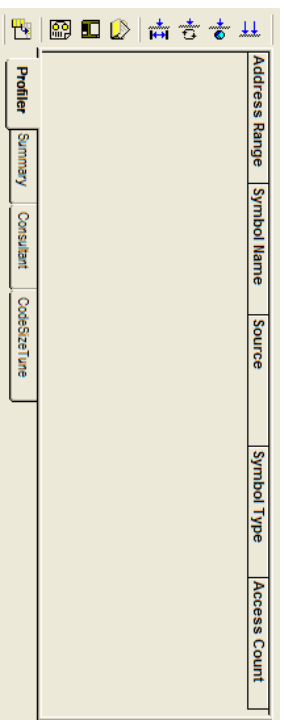
26/09/2007



This causes the profiler to collect data these activities and update the Profile Viewer every two seconds.

19. Right-click on the Profile Setup tool and click Hide to close the window.

20. Scroll down close to the bottom of the Setup Advice page in the Advice Window. In the Action box, click on the link to the Profile Viewer. The Profile Viewer will open. This tool can also be opened using the Profile->Viewer menu item.



file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007

Getting Started With the Code Composer Studio Tutorial

Page 168 sur 548



### Default Data View Description

DASH - L4

1. Run the modern application by pressing F5, or by selecting Run from the Debug menu. The selected functions and ranges will appear in the Profile Viewer. The profiler will update the data values for each item every two seconds.
2. After the functions and updated data appear, halt the application by pressing Shift-F5, or by selecting Halt from the Debug menu.
3. Scroll to the right of the Profile Viewer to view all of the data columns. Your numbers may vary.

Address Range	Symbol Name	SLR	Symbol Type	Access Count	cycle: Total	In...	cycle: Total	Ex...	CPU:discontin...	CPU:discontin...	CPU:instruct...	CPU:instruct...
0x01220-0x1210	AddItoSignal	274-301.modemtx.c	function	0	0	0	0	0	0	0	0	0
0x0160-0x11f0	Initialize	204-248.modemtx.c	function	1	316	322	0	0	6	508	514	0
0x012f0-0x160c	ShapimgFilter	121-146.modemtx.c	function	0	0	0	0	0	0	0	0	0
0x011f0-0x1220	Simelookup	54-70.modemtx.c	function	32	416	608	0	0	192	300	492	0
0x0160c-0x16ec	main	83-329.modemtx.c	function	1	1588	9	0	0	0	1180	6	0
0x0122c-0x12f0		280-300.modemtx.c	range	0	0	0	0	0	0	0	0	0




The data set includes the following columns for the previously selected events:

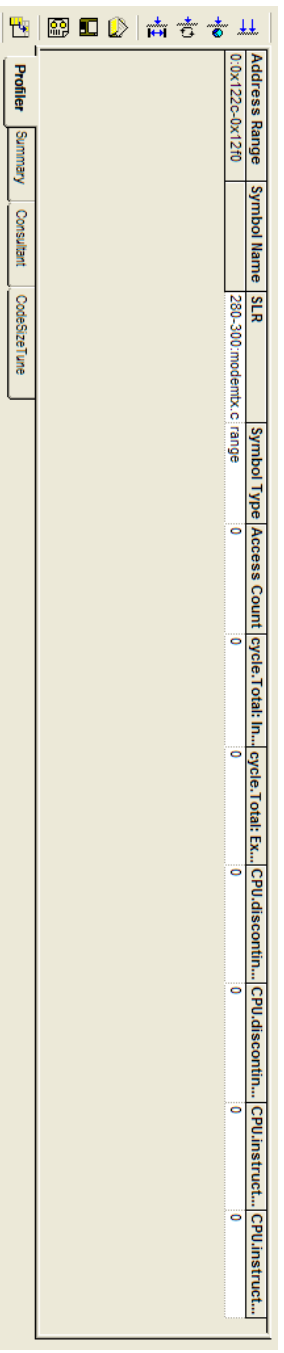
- Address Range displays the hexadecimal range of the profiled section of code.
- Symbol Name contains the name of the function. If the address range is a function.
- SLR contains the line numbers and file name of the profiled code.
- Symbol Type displays the type of range (function, loop, or range) of the profiled section of the code.
- Access Count shows the number of times the profiled section of code was entered the last time the program was run.
- cycle:Total: Incl. Total displays the number of cycles that occurred in the entire profiled section of code, including subroutines. This column is included in the data set because

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007

selecting the "Collect application level profile for total cycles and code size" automatically selects the cycle→Total event.

- cycle>Total: Excl. Total shows the number of cycles that occurred in the profiled section of code, excluding subroutines. The data set includes this column because we selected "Collect application level profile for total cycles and code size," which automatically selects the cycle→Total event.
  - CPU discontinuity/interrupt.summary: Incl. Total displays the number of CPU interrupts that occurred in the entire profiled section of code, including subroutines. The data set includes this column because we selected the CPU→discontinuity→interrupt event in the [Profile Setup](#) lesson.
  - CPU discontinuity/interrupt.summary: Excl. Total shows the number of CPU interrupts that occurred in the profiled section of code, ignoring subroutines. The data set includes this column because we selected the CPU→discontinuity→interrupt event in the [Profile Setup](#) lesson.
  - CPU instruction-executed: Incl. Total displays the number of CPU instructions that were executed in the profiled section of code, including subroutines. The data set includes this column because we selected the CPU→instruction→executed event in the [Profile Setup](#) lesson.
  - CPU instruction-executed: Excl. Total displays the number of CPU instructions executed in the profiled section of code, ignoring subroutines. The data set includes this column because we selected the CPU→instruction→executed event in the [Profile Setup](#) lesson.
4. You can filter the data view of profiled code ranges according to symbol type. Click on the Show Functions Only icon  in the Profile Viewer. This will remove all items other than functions from the view.
  5. Click on the Show Loops Only icon  in the Profile Viewer. Only profiled loops will remain in the data view. In this example, we did not choose any loops for profiling, so the view should be empty.
  6. Click on the Show Ranges Only icon  in the Profile Viewer. The profiled ranges will remain in the data view.



Address Range	Symbol Name	SLR	Symbol Type	Access Count	cycle:Total	In...	cycle:Total	Ex...	CPU:discontin...	CPU:discontin...	CPU:instruct...	CPU:instruct...
0:0x122c-0x1210	280_300_modemtk_c_range			0	0	0	0	0	0	0	0	0

7. Click on the Show all Profile Items icon  in the Profile Viewer. This displays all functions, loops, and ranges in the data view.

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007


Getting Started With the Code Composer Studio Tutorial

Page 170 sur 548



**Customize Data Views**

DASH - L4

1. In order to view the most relevant tuning information in the Profile Viewer, it may be necessary to customize the data view. Columns and rows can easily be added, removed, or rearranged. Select the row titled AddNoisesignal by clicking on its address in the Address Range or Symbol Name column.
2. Right-click anywhere on the Profile Viewer and click Hide Selection. This removes the entire row from the data view.
3. Click on the Columns and Rows Setting icon . The Columns and Rows Setting window will appear. Your entries may vary.



The window shows the hidden and visible rows and columns. It provides a way to add and remove both rows and columns in the data view.

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007

4. In the list of visible columns, select the column titled CPU.Instruction.Executed: Excl. Total.
5. Click on the double-back-arrow button (<<) to move the selected column to the list of hidden columns.
6. Click OK to remove the column.
7. Click on the CodeSizeTune tab in the Profile Viewer. With the exception of the Profiler tab, Profile Viewer tabs display only data that can be tuned by the corresponding tool. Thus, the CodeSizeTune tab contains only the data that can be tuned by CodeSizeTune.
8. Return to the Profiler tab. Highlight the fourth-last row in the data view. Hold down the Shift key and highlight the last row in the view to select the last four rows. Your numbers may vary.

Address Range	Symbol Name	SLR	Symbol Type	Access Count	cycle:Total	In...	cycle:Total	Ex...	CPU.discontin...	CPU.discontin...	CPU.discontin...	CPU.instruct...
0x01f0-0x11f0	Initialize	204-248.modemtx.c	function	1	316	0	322	0	6	0	0	508
0x012f0-0x160c	ShapingFilter	121-148.modemtx.c	function	0	416	0	608	0	192	0	300	1180
0x011f0-0x1220	ShapingFilter	83-329.modemtx.c	function	32	1588	9	1588	0	0	0	1180	0
0x0160c-0x160c	main	280-300.modemtx.c	range	0	0	0	0	0	0	0	0	0

9. Drag the selected rows to the first position in the data view before releasing the mouse button. The rows will be relocated to the top of the view. The Profile Viewer provides the ability to move individual rows or groups of rows to any location in the data view.
10. Columns can be arranged in the same manner. Click on the top of the Symbol Type column to select the entire column. Drag and drop the column to another location in the data view. The view will be updated with the column in the new location.

## Sorting Data

DASH - L4

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007

Getting Started With the Code Composer Studio Tutorial

Page 172 sur 548

1. The Profile Viewer can easily pinpoint which profiled sections of code require the most tuning. For example, select the entire cycle.Total: Excl. Total column. Double-click on the title of the column. The contents of the column should now be sorted from largest to smallest. Your numbers may vary.


Address Range	Symbol Name	Symbol Type	SLR	Access Count	cycle:Total	In...	cycle:Total	Ex...	CPU.discontin...	CPU.discontin...	CPU.discontin...	CPU.instruct...
0x011f0-0x1220	SineLCoUp	function	54-70.modemtx.c	32	416	0	608	0	192	0	300	508
0x0160-0x11f0	Initialize	function	204-248.modemtx.c	1	316	0	322	0	6	0	0	1180
0x0160c-0x160c	main	function	83-329.modemtx.c	1	1588	9	1588	0	0	0	1180	0
0x012f0-0x160c	ShapingFilter	function	121-148.modemtx.c	0	0	0	0	0	0	0	0	0
0x0122c-0x12f0		range	280-300.modemtx.c	0	0	0	0	0	0	0	0	0

The function, loop, or range with the largest number of cycles is possibly a good place to focus tuning efforts. The Divide-and-Conquer tuning strategy suggests dividing the functions into smaller profile ranges to determine which section of code uses the most cycles. For more information on the Divide-and-Conquer tuning strategy, click on the following link:

2. The data can also be sorted from smallest to largest. Select the column and double-click it once more to re-sort the data.


## Saving and Restoring Data Sets

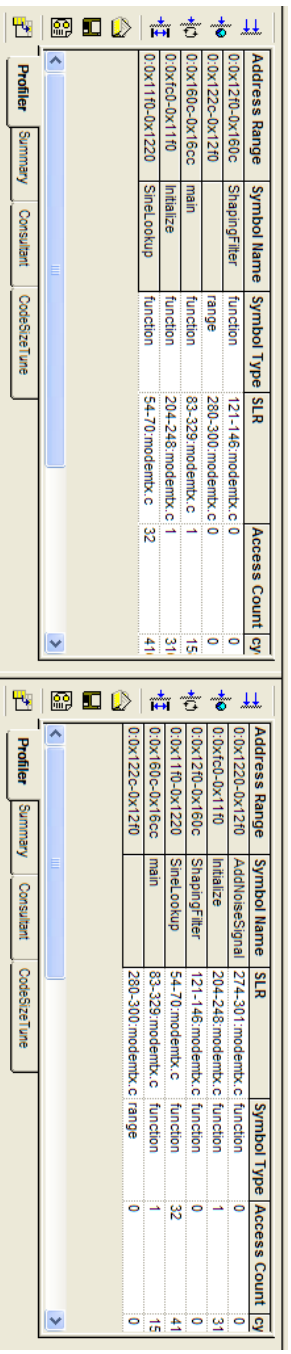
DASH - L4

1. In order to compare data collected from two or more executions of an application, the current data set must be saved in the Profile Viewer. Click on the Save Current Data Set icon  in the Profile Viewer window. The Save As window will appear.
2. Browse to the location of the debug executable for the project, which should be C:\CCStudio\_v3.10\tutorial\sim64xx(modem)\Debug.
3. Enter a name for the data set such as Initial\_run.xml.
4. Click the Save button to save the data set for reopening later.

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007


- You can also save data in a text file with one row on each line and columns separated by commas. These files are saved with a .csv extension and can be opened in various spreadsheet applications for further analysis. To save a file in this format, use the Export to Delimited Text File icon .
- Click on the Debug→Restart menu item to reset the data collection totals.
- In the Setup tab of the Advice Window, click on the Profile Data Viewer link to launch a second Profile Viewer window, so that we can compare the two data sets. Your numbers may vary.

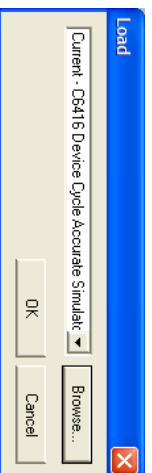


Address Range	Symbol Name	Symbol Type	SLR	Access Count	cy
0x01270-0x160c	ShapingFilter	function	121-146:modemtx_c	0	0
0x0122c-0x1270	range	range	280-300:modemtx_c	0	0
0x0160c-0x160c	main	function	83-329:modemtx_c	1	31
0x0160-0x1170	Initialize	function	204-248:modemtx_c	0	0
0x011f0-0x1220	Shelookup	function	54-70:modemtx_c	32	41
			54-70:modemtx_c	1	15
			32	41	15

Address Range	Symbol Name	SLR	Symbol Type	Access Count	cy
0x01220-0x1270	AddNoiseSignal	274-301:modemtx_c	function	0	0
0x01c0-0x1170	Initialize	204-248:modemtx_c	function	1	31
0x01270-0x160c	ShapingFilter	121-146:modemtx_c	function	0	0
0x011f0-0x1220	Shelookup	54-70:modemtx_c	function	32	41
0x0160c-0x160c	main	83-329:modemtx_c	function	1	15
0x0122c-0x1270	range	280-300:modemtx_c	range	0	0

- Press F5 to run the application, or select Run from the Debug menu. Both Profile Viewers will collect data for the running application.
- After the functions appear and the data has been updated, halt the application by pressing Shift-F5, or by selecting Halt from the Debug menu.
- Click on the Load Data Set icon  in the second Profile Viewer.



- When the Load window appears, click on the Browse button.
- When the Open window appears, browse to the location of the saved data set (initial\_run.xml file) and double-click on the file.

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007

Getting Started With the Code Composer Studio Tutorial

Page 174 sur 548

- Click OK to restore the data set in the Profile Viewer. With both data sets open side-by-side, compare the new and old profiling data. This will help to determine where tuning improvements have been made at the function and loop level.

- Close both Profile Viewer windows by right-clicking each window and clicking on Close.

The next lesson in the Tuning Dashboard tutorial will use a sample project to introduce the [tuning process](#) using all the tools we have reviewed. Be sure to un-check the Activities boxes, delete any functions or ranges, and remove all control points before moving to the next lesson.



### Tuning an Application Overview

DASH - LS

#### Learning Objectives:

- Choose activities by utilizing Profile Setup
- Set cycle count and code size application goals
- View the data collected for specific activities in the Profile Viewer

#### Example used in this lesson: Matrix (mat\_oprn1)

**Target Configuration:** The mat\_oprn demonstration application is located in the C:\CCStudio\_V3.10\tutorial\sim64xxx\mat\_oprn\mat\_oprn\_1 directory. To run this tutorial, you must setup the Code Composer Studio™ IDE to run on a 6416 device cycle accurate simulator, little endian. For more information about Setup, run Setup using the Setup CCS icon, and use the online help.

#### Application Objective:

This tutorial incorporates the tools used in the previous lessons to tune a sample application.

This lesson consists of the following steps. To go directly to a step, click on the link below:

[Choose Activities](#)

[Set Project Goals](#)

[View Collected Data](#)




file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007

## Choose Activities

DASH - L5

To tune our project, we will first load it and choose the profiling activities using profile setup.

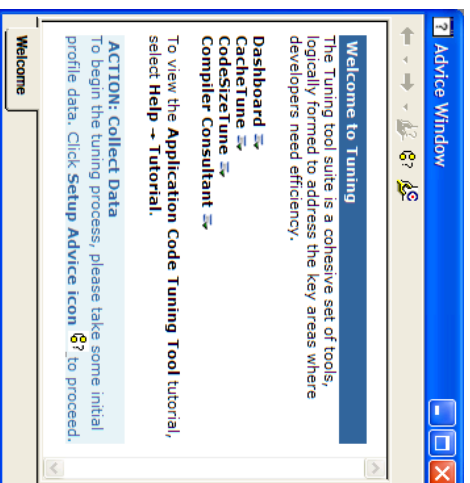
1. If you have not already done so, from the Windows Start menu, choose Programs → Texas Instruments → Code Composer Studio 3.1 → Code Composer Studio 3.1. (Or, double-click the Code Composer Studio icon on your desktop.)
2. From the Project menu, choose Open, and browse to: `C:\CCStudio_v3.10\tutorial\sim64xx\mat_oprn\mat_oprn_1`
3. Double-click on `mat_oprn_1.pjt` to open the project. This will open the example project used in this lesson.
4. Under the Project menu, choose Build Options...
5. In the Build Options dialog, click the Compiler tab.
6. Verify that the Generate Debug Info option has Full Symbolic Debug (-g) selected from the drop-down list, and click OK.
7. From the Project menu, choose Rebuild All. An output file (.out) must be generated and loaded before beginning the tuning process.
8. From the File menu, choose Load Program.
9. Browse to the location of the output file, which should be: `C:\CCStudio_v3.10\tutorial\sim64xx\mat_oprn\mat_oprn_1\Release`.
10. Select `mat_oprn_1.out` and click Open. When the example program has been loaded, the Disassembly window will open in the editor.
11. Activate tuning mode by clicking on the tuning fork icon  or by clicking the menu item View → Layout → Tuning Layout. Switching to Tuning mode reorganizes the Code Composer Studio workspace to provide a good view of both the Project and the Advice Window, which helps to guide the tuning process. You can float this window in your main workspace by right-clicking on the window and choosing Float in main window from the context menu.



file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007

Getting Started With the Code Composer Studio Tutorial

Page 176 sur 548

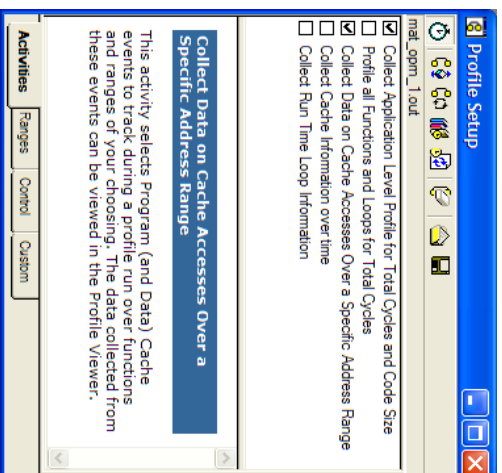


12. In the Advice window, click on the Setup Advice icon . The Setup tab will open in the Advice Window.
13. In the first Action box of the Setup tab, click on the link to launch the Profile Setup tool. The Profile Setup tool will open on the right side of Code Composer Studio. This tool can also be opened using the Profile—Setup menu item.
14. The Profile Setup allows the setting of several options for data configuration. You can float this window in your main workspace by right-clicking on the window and choosing Float in main window from the context menu.
15. Check the icons at the top of the Profile Setup tool to determine if profiling is enabled. The Enable/Disable Profiling icon  must be toggled on to enable profiling. Profiling may not be enabled if the program has started running or if it was explicitly disabled. Data will not be collected if profiling is not enabled.
  - Collect Application Level Profile for Total Cycles and Code Size
  - Collect Program Cache Information over time {CG20X (not CG211), CG701}
  - Collect Data on Cache Accesses Over a Specific Address Range {All Other 6x ISAs}
16. As the Advice Window indicates, profiling must be set up before data can be collected. In the Activities tab of the Profile Setup tool, select the following items. Please see the [Profile Setup](#) lesson for more details:
  - Collect Application Level Profile for Total Cycles and Code Size
  - Collect Program Cache Information over time {CG20X (not CG211), CG701}
  - Collect Data on Cache Accesses Over a Specific Address Range {All Other 6x ISAs}

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007





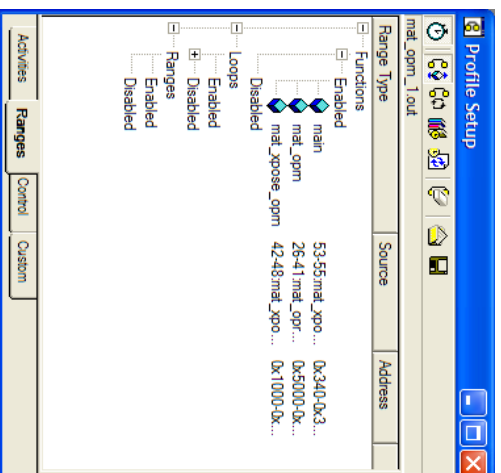
17. In the Ranges tab of the Profile Setup tool, click on the Enable/Disable All Functions icon . This will add every function in the Matrix application to the Functions branch and collect data upon execution. If any functions are disabled, click on them and press the space key on your keyboard to move them to the enabled list.

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007

Getting Started With the Code Composer Studio Tutorial

Page 178 sur 548



18. Since the application will always terminate, it does not require an exit point.
19. Click on the Custom tab and select the following Range items for these ISAs only: C55x, C620x, and C6701:
- Program\_cache->miss->summary
  - cycle->Total

20. Close the Profile Setup tool by right-clicking on it and selecting Hide.

In the next lesson, we will set project goals for code size and cycle count.



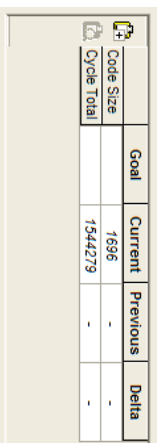
**Set Project Goals**

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007

In this lesson, we will set project goals for code size and cycle count using the goals window.

1. Press F5 to run the application, or choose Run from the Debug menu. The output window will display any output from the application.
2. If the General Advice tab does not open automatically in the Advice Window, click on the View General Tuning Advice icon . This opens the General tab that contains the Setting Goals section.
3. If the Goals window does not open automatically, click on the Launch the Goals Window icon . In the Action box in the Setting Goals section of the General Advice window, the Goals Window will open on the left side of Code Composer Studio. You may also launch it by clicking on Profile→Tuning→Goals. Your numbers will vary.



Goal	Current	Previous	Delta
Code Size	1696	-	-
Cycle Total	1544279	-	-

The Goals Window will contain the initial set of data. These values represent the Code Size and Cycle Total. The Code Size is the total number of bytes required to store the program instructions. The Cycle Total is the total number of cycles required to execute the instructions in the application, including both memory accesses and input/output operators, when applicable. Your values may vary.

4. For the purposes of this example, enter a larger value in the Goal column for Code Size than the Current Code Size. Since the goal has been met, the Current value will change color to green and appear in parentheses.
5. Enter a value in the Goal column for Cycle Total that is less than the Current Cycle Total. The current value will change to red because the goal has not been met.



Goal	Current	Previous	Delta
Code Size	1796	(1696)	-
Cycle Total	1544179	1544279	-

6. Select the Debug→Reset CPU menu item, then the File→Reload Program menu item to position the Program Counter at the beginning of the application. This will also reset the cache. In the next lesson, we will view the collected data using the profile viewer.

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007

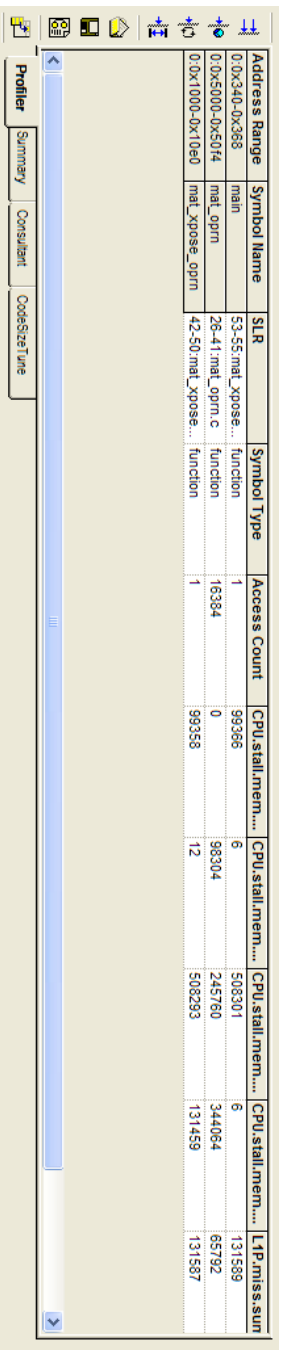
Getting Started With the Code Composer Studio Tutorial

Page 180 sur 548

## View Collected Data

In this lesson, we will view the data we collected for the specific activities using the profile viewer.

1. Click on the Setup tab in the Advice Window to display the Setup advice page.
2. In the Action box at the bottom of the page, click on the link to launch the Profile Viewer. The Profile Viewer will open at the bottom of the workspace. If necessary, resize the Profile Viewer to view as many columns as possible. You can also launch this tool from the Profile→Viewer menu.
3. Press F5 to run the application, or choose Run from the Debug menu. Both the Goals Window and the Profile Viewer will be updated with new data for the application. Your numbers may vary.



Address Range	Symbol Name	SLR	Symbol Type	Access Count	CPU stall mem....	CPU stall mem....	CPU stall mem....	CPU stall mem....	L1Pmiss, sur
0x0340-0x388	main	53-55:mat_xpose...	function	1	99386	6	508301	6	131589
0x4500-0x50f4	mat_oprn	28-41:mat_oprn.c	function	16384	0	98304	245760	344064	65792
0x01000-0x10e0	mat_xpose_oprn	42-50:mat_xpose...	function	1	99358	12	508293	131459	131587

4. As we selected different activities for different ISAs, the columns in the Profile Viewer will also be different. For all 6x ISAs (except C620x and C6701), select the CPU stall mem.L1D: Incl. Total column. This column counts the total number of stalls that occur when accessing L1D data cache. Double-click this column to sort it in descending order. The function that causes the most data stalls appears at the top of the sorted data set.


5. Again for all 6x ISAs (except C620x and C6701), select the CPU stall mem.L1P: Incl. Total column. This data value represents the total number of stalls that occur when accessing L1P program cache. Double-click this column twice to sort it in descending order. Again, the function that causes the most data stalls appears at the top of the sorted data set. This function and the one in the previous step are prime candidates for cache tuning.

6. Specifically for the C55x, C620x, and C6701 ISAs, select the Program cache miss summary: Incl. Total column. This column counts the total number of cache misses that occur when accessing program cache. Double-click this column twice to sort it in descending order. The function that causes the most data misses appears at the top of the sorted data set. This

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007

function is a prime candidate for cache tuning.

- The current data set must be saved in the Profile Viewer so that it can be compared with future data if desired. Click on the Save Current Data Set icon  in the Profile Viewer window. The Save As window will appear.
- Browse to the location of the debug executable for the project, which should be `C:\CCStudio_v3.1.0\tutorial\sim64xx\mat_oprn\mat_oprn_1\Release`. Enter a name for the data set, such as `initial_run.xml`. Click the Save button to save the data set for later.
- At this point, the CacheTune tool may be used to decrease the number of cache misses in the Matrix application. Please see the [CacheTune tutorial](#) for more information.

At this point we have introduced the basic pieces of the dashboard (profile setup, goals window, advice window and profile viewer) and have demonstrated how they can be used to tune an application. This concludes the Dashboard tutorial.



## CodeSizeTune Tutorial Overview

### CST - Intro

CodeSizeTune (CST) is a tool that enables you to more quickly and easily optimize the trade-off between code size and cycle count for your application. Using a variety of profiling configurations, CodeSizeTune will profile your application, collect data on individual functions, and determine the best combinations of compiler options. CST will then produce a graph of these function-specific options sets, allowing you to graphically choose the configuration that best fits your needs.

Before using this tutorial module, you should have done the following:

- Installed Code Composer Studio™ IDE
- Set up the Code Composer Studio™ IDE to run on the C6416 device cycle accurate simulator, little endian.

We have provided two different tutorial lessons, depending on your time availability. The Quick Start tutorial will quickly introduce the main aspects of the tool by profiling a simple example.

The In-Depth tutorial will look at the many features and capabilities of CST, using a more complex example.

**Target Configuration:** To run these tutorials, you must set up the Code Composer Studio™ IDE to run on the C6416 device cycle accurate simulator, little endian. The CodeSizeTune tool will function on all 6x simulators. For more information on the Setup utility, access the [Configuring Target Devices](#) tutorial or the online help.

This tutorial module contains the following lessons:

#### Quick Start Tutorial

#### In-Depth Tutorial



file:///C:/Documents and Settings/Pierre-Armand/Local Settings/Temp/~hhBA53.htm

26/09/2007

## Getting Started With the Code Composer Studio Tutorial

Page 182 sur 548

The forward arrow will take you to the next page in this lesson.

### Overview

CST - LI

In this lesson, you will use a step-by-step process to optimize an application using CodeSizeTune.

Learning Objectives:

- Prepare the application for profiling
- Profile the application with CodeSizeTune
- Analyze the CST output graph
- Determine the best function-specific options set
- Save a setting as a configuration

#### Example: modem

Application Objective:

This lesson demonstrates several features of the CodeSizeTune Tool. You will learn how to set up your application for profiling using exit points. Code Size Tune will profile the application for you, and recommend various options sets. Then you will analyze the sets using an output graph and determine which one would work best for your application. Finally, you will save the setting for future use. The Modem demonstration is an implementation of a typical modem transmitter system. Data is read in 1 baud, then processed into an in-phase (I) and quadrature (Q) component using a raised cosine filter (ShapingFilter() function). Resultant I and Q data are modulated (Via Modulation() function) and added together to generate a transmitter output.

This lesson consists of the following steps. To go directly to a step, click on the link below:

[Launching CodeSizeTune](#)

[Using Exit Points](#)

[Building and Profiling](#)

[Viewing and Selecting Function-Specific Options Sets](#)

[Saving a Configuration](#)



file:///C:/Documents and Settings/Pierre-Armand/Local Settings/Temp/~hhBA53.htm

26/09/2007

The forward arrow will take you to the next page in this lesson.

### Launching CodesizeTune

CST - L1

This lesson opens the Modern demonstration application.

Over the course of the lessons, you will utilize the following types of files:

- .c This file contains source code that provides the main functionality of this project
  - .jit This file contains all of your project build and collection option sets
1. If you have not already done so, from the Windows Start menu, choose Programs→Texas Instruments→Code Composer Studio 3.1→Code Composer Studio 3.1. (Or, double-click the Code Composer Studio icon on your desktop.)
  2. From the Project menu, choose Open.
  3. Browse to [C:\CCStudio\\_v3.10\Tutorial\sim64xx\Modern\](#)
  4. Select modern.pjt and click Open to load the project.
  5. Under the Project menu, choose Build Options...
  6. In the Build Options dialog, click the Compiler tab.
  7. In the scrollable window at the top of the dialog, replace the -g option with -gp, and click OK to save.

#### Before:

```
-g -q -o3 -fr"C:\CCStudio_v3.10\Tutorial\sim64xx\modern\Debug" -mv6400
```

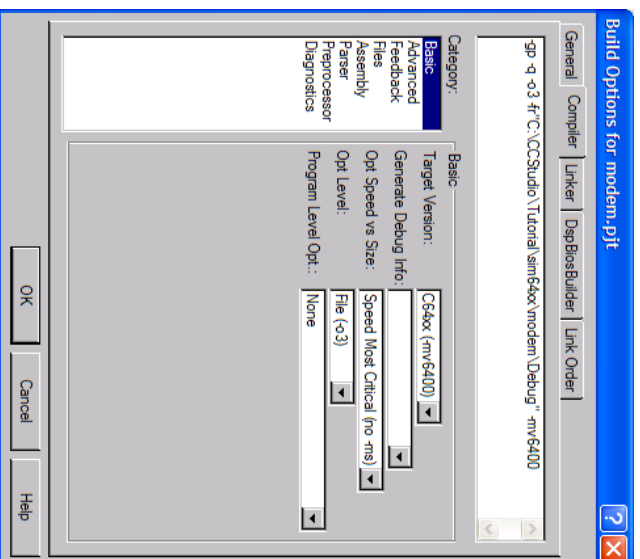
#### After:

[file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm](#)

26/09/2007

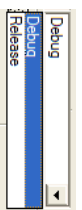
### Getting Started With the Code Composer Studio Tutorial

Page 184 sur 548



Note: -gp will allow you to set an exit point at the end of the loop in the next lesson.

8. Verify that the output file will be generated into the Debug folder in the project drop-down menu.

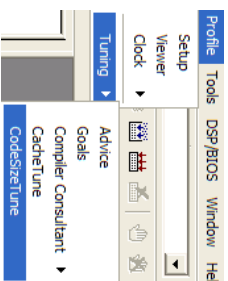


9. From the Project menu, click on Rebuild All.

[file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm](#)

26/09/2007

- From the File menu, click on Load Program, and choose modem.out from the Debug folder.
- Launch CodeSizeTune from the Profile->Tuning menu by choosing CodeSizeTune.



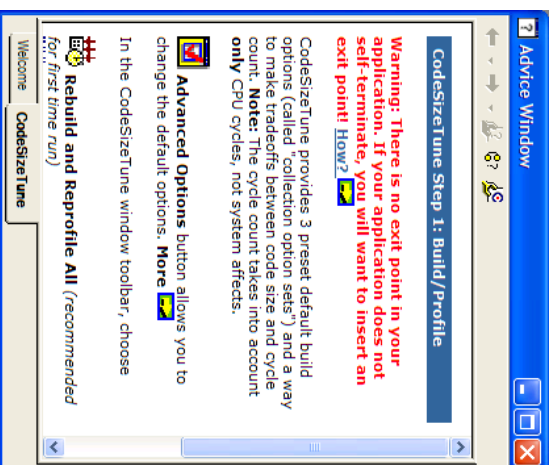
Once CodeSizeTune is launched, the CodeSizeTune window displays. Also, the advice window displays the initial CodeSizeTune advice topic. You can float this window in your main workspace by right-clicking on the window and choosing Float in main window from the context menu. You can also resize it, as necessary.

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007

Getting Started With the Code Composer Studio Tutorial

Page 186 sur 548



In the next lesson, we will set an exit point in the modem file.



### Setting Exit Points

CST - L1

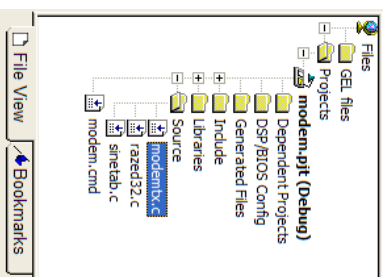
In addition to other preliminary requirements, discussed in the CodeSizeTune online help files, your application must self terminate. The Modem application used in this tutorial is a looping application that does not self-terminate. In order for CST to work, we must force it to stop data collection at some point by establishing an exit point. For CST, the exit point will also halt the CPU.


In this example, we will tell CodeSizeTune to stop profiling after the function main by inserting an exit point at the end of the loop, and indicating that we would like CodeSizeTune to ignore that exit point three times before exiting.

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007

1. Click on the modem.plt project in the Project View Window to open the project file list.
2. Click on the Source folder inside Modem.plt. Navigate to the modemx.c and double-click to open it.



3. From the Profile menu, choose Setup.
4. Check the icons at the top of the Profile Setup tool to determine if profiling is enabled. The Enable/Disable Profiling icon  must be toggled on to enable profiling. Profiling may not be enabled if the program has started running or if it was explicitly disabled. Data will not be collected if profiling is not enabled.
5. Click on the Control tab of the Profile Setup window. Your line number may vary.
6. Navigate to the open modemx.c window. Navigate to the while function loop within the main function of modemx.c, and highlight the end brace ("}") on line 344 of the [while loop](#).
6. Drag-and-drop it to the Exit Point pane within the Control tab. This adds and displays an exit point in the Exit Point pane. The default count is set to 1.
7. Click on the count field and change the count to 3.

In the next lesson, we will build and profile our application.

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007


Getting Started With the Code Composer Studio Tutorial

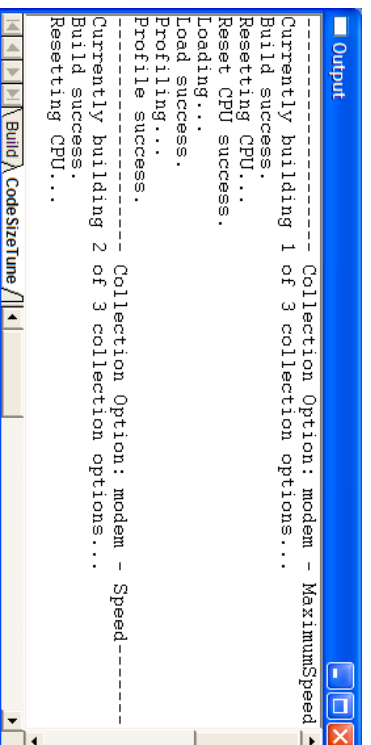
Page 188 sur 548

## Building and Profiling

CST - LI

Now that we have prepared our application by including an exit point, we are ready to build and profile. The first time that you run CodeSizeTune on an application, you should do a Rebuild and Reprofile All to set up the system.

1. In the CodeSizeTune window tool bar, choose  RebuildAll and ReprofileAll.
2. CodeSizeTune profiles the Modem application using three default collection options sets: MaximumSpeed, MinimumSize, and Speed. If you would like to learn more about these default options, click the link on the Advice Page.
3. The CodeSizeTune tab shows build progress in the output window, first building, resetting the CPU, then loading, and finally profiling for each of the preset collection options sets.



When CodeSizeTune finishes profiling your application under each collection option, it displays the output graph.

In the next lesson, we will analyze the different options sets displayed on the output graph.

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007

**Viewing and Selecting Function-Specific Option Sets**

CST - L1

Once CodeSizeTune has built and profiled all active collection options sets, it automatically displays the graph. The CodeSizeTune graph plots function-specific option sets according to code size and cycle count. Code size is plotted on the x-axis and cycle count on the y-axis. We will now explore the different tools in the output window.

**Note:** If the CodeSizeTune output window graph does not display after you have profiled, check to see if another window has hidden the output window in the workspace.

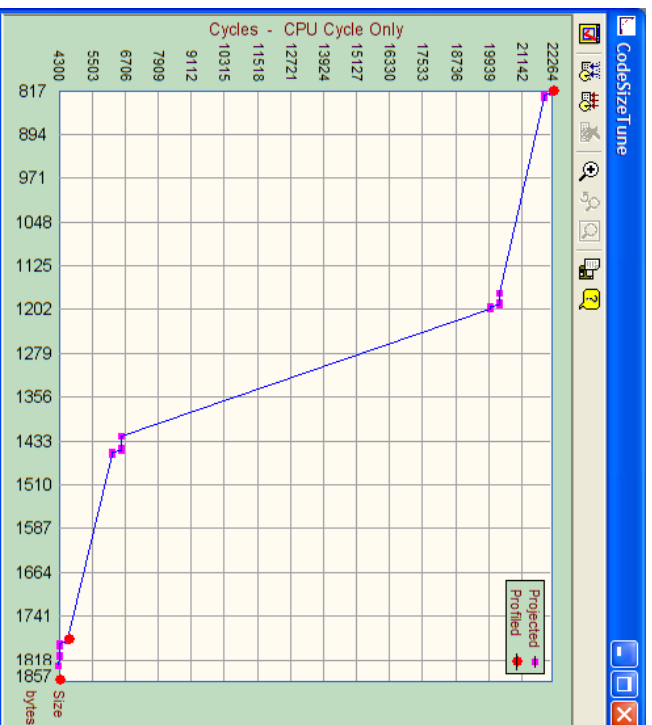
1. Hold the mouse pointer over any of the points to display its code size and cycle-count information. Let's take a closer look at the graph and choose a function-specific options set. Your graph may vary.



file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007

Getting Started With the Code Composer Studio Tutorial


Page 190 sur 548



2. Select Zoom (  ) from the menu.
  3. Left-click to drag a border around the area of the curve you would like to examine more closely.
  4. Continue this zooming process until you are satisfied with the level of detail.
- If you would like to go back to the last level of zoom, click on the Undo-Zoom  button.

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007

5. Deselect Zoom () to return your cursor to its normal appearance.
6. Pick one of the points (in purple) with the desired cycle and code size, and click on it to make it the Selected Function-Specific Options Set. The Profile Viewer window automatically displays when you select a point from the graph.
7. The Profile Viewer window displays detailed information about the selected function-specific options set. This information includes the size, cycle, name and options for each function. The comprehensive tab includes all profile data information. To see specific information for CodeSizeTune, select the CodeSizeTune tab.

The next lesson will teach you how to save this setting as a collection option set.



### Saving a Configuration

CST - L1

The last step you need to perform is saving the selected options set (the single graph point) as a Code Composer Studio™ project configuration.

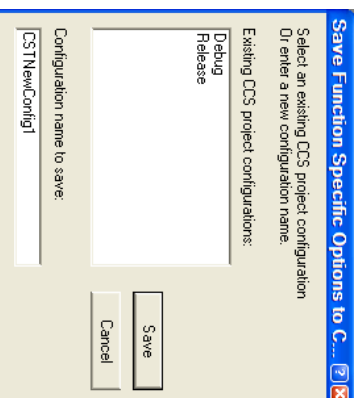
1. Click the Save () button from the CodeSizeTune toolbar.
2. Type the name of the new configuration, and click Save. The default name is CSTNewConfig1. Code Composer Studio will automatically use this just saved configuration.

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007

Getting Started With the Code Composer Studio Tutorial

Page 192 sur 548



3. To view this configuration, select this configuration from the Project Configuration drop-down menu. Or you can open the Project menu, select Configurations... and choose CSTNewConfig1.



4. From the Project menu, choose Rebuild All. This action rebuilds the project using the new project configuration, which contains the function level build options defined by CodeSizeTune. The .out file is built into the C:\CCStudio\_v3.10\tutorial\sm64xx/modem\yourname directory.
  5. To reset the configuration back to Debug, choose Debug from the Project Configuration drop-down menu.
- After saving your selected settings to a Code Composer Studio™ project, you can now continue to build your application, knowing that the optimal function-level set of compiler options are being used for your given speed and size constraints.

This concludes the CST quick start tutorial. To learn about the full range of CST's capabilities, go on to the [In-Depth](#) tutorial.



### Overview

CST - L2

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007



This tutorial takes you through a step by step process for optimizing an application using CodeSizeTune (CST).

Be aware before you begin that it may take some time to build and profile multiple collection option sets, so read through the tutorial first and plan ahead. The actual speed may vary depending on the host machine.

Learning Objectives:

- Prepare your application for profiling
- Analyze collection options sets
- Profile your application with CodeSizeTune
- Analyze the CodeSizeTune output graph
- View and apply overrides
- Save settings as configurations

Example: ZLIB

Target Configuration: The ZLIB demonstration application is located in the `C:\CCStudio_v3.10\tutorial\sim64xx\zlibdemo` directory. To run this tutorial, you must set up the Code Composer Studio™ IDE to run on the C6416 device accurate simulator, `litle.endian`. For more information about Setup, run Setup using the Setup CCS icon, and use the online help.

Application Objective:

This lesson demonstrates many of the features of the CodeSizeTune Tool. You will learn how to set up your application for profiling using halt and resume points. CodeSizeTune will profile the application for you, and recommend various options sets, depending on the parameters that you set. Then you will analyze the sets using an output graph and determine which one would work best for your application. Finally, you will save the setting for future use. The ZLIB demonstration application is an implementation of the Zlib compression and decompression algorithm. ZLIB reads in a hard-coded buffer, compresses, and then decompresses the buffer, and compares the original buffer with the decompressed buffer for correctness.

This lesson consists of the following steps. To go directly to a step, click on the link below:

[Launching CodeSizeTune](#)

[Using Halt Collection and Resume Collection Points](#)

[Selecting Collection Options Sets](#)

[Building and Profiling](#)

[Viewing and Selecting Function-Specific Options Sets](#)

[file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm](#)

26/09/2007

Getting Started With the Code Composer Studio Tutorial

Page 194 sur 548

[Viewing Overrides](#)

[Selecting Rules and Overrides](#)

[Saving a Configuration](#)



The forward arrow will take you to the next page in this lesson.

### Launching CodeSizeTune

CST - L2

This lesson uses the ZLIB demonstration application. Before you use CodeSizeTune, you must open up the ZLIB project and make sure it is set up for profiling with CodeSizeTune.

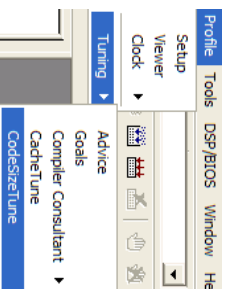
Over the course of the lessons, you will utilize the following types of files:

- `.c` This file contains source code that provides the main functionality of this project
- `.jit` This file contains all of your project build and collection option sets

1. If you have not already done so, from the Windows Start menu, choose Programs→Texas Instruments→Code Composer Studio 3.1→Code Composer Studio 3.1. (Or, double-click the Code Composer Studio icon on your desktop.)
2. From the Project menu, choose Open.
3. Browse to `C:\CCStudio_v3.10\tutorial\sim64xx\zlibdemo`
4. Select `Zlib0.jit` and click Open.
5. From the Project menu, click on Rebuild All.
6. From the File menu, click on Load Program, and choose `zlib32.out` from the Debug folder. Click Open.
7. Click on the Profile menu and choose Tuning, and then CodeSizeTune to enable CodeSizeTune.

[file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm](#)

26/09/2007



CodeSizeTune assumes that your application is setup for profiling, and therefore expects your application to:

- Run as a batch program (non-interactive with the user).
- Self terminate or have an Exit Point established. For more information see the CodeSizeTune online help topic on Using Exit Points.
- Use input that is representative of typical usage. Input can be data, interrupts, and other stimuli that affect your application.
- Use identical input for each invocation of the application.

The CodeSizeTune Advice page provides useful links and access to information on topics as it progresses through the stages of profiling for CodeSizeTune. Once CodeSizeTune has been invoked, the Advice window will display the initial CodeSizeTune Advice topic.



### Using Halt and Resume Collection Points

CST - L2

If you find that you would like to profile only certain portions of your application, you can use Halt Collection/ Resume Collection points. These allow you to set a profile range within your source code. When CodeSizeTune encounters a Halt Collection point, it will no longer collect profile information. When it subsequently encounters a Resume Collection point, it will resume the collection of profile information. To read more about how to set and modify profile ranges through the use of Halt Collection and Resume Collection points, please refer to the related tutorial on [Profile Setup](#).


In our example, we are going to narrow the profile range within an application using Halt Collection and Resume Collection points. The source file `zlib.c` includes the functions `do compress` and `do decompress`, and these two functions are the most critical for profiling. A Resume Collection point at the beginning of the `do compress` function and a Halt Collection point at the end of the `do decompress` function will define the range for CodeSizeTune to profile.

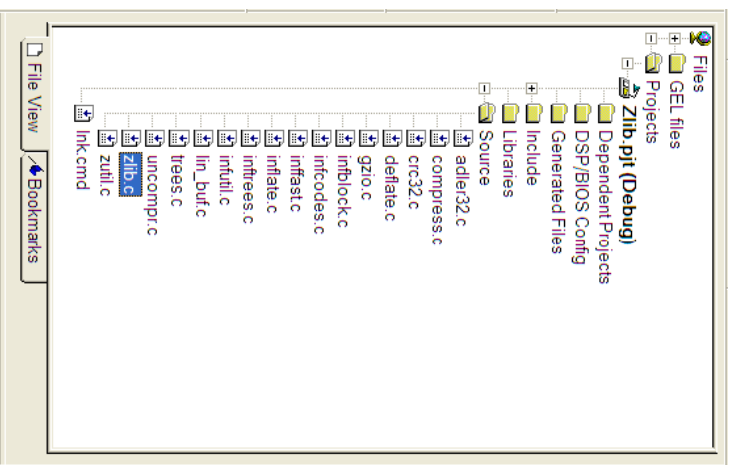
<file:///C:/Documents and Settings/Pierre-Armand/Local Settings/Temp/~hhBA53.htm>

26/09/2007

### Getting Started With the Code Composer Studio Tutorial

Page 196 sur 548

1. Click on the Profile menu, and choose Setup to open the Profile Setup window.
2. Check the icons at the top of the Profile Setup tool to determine if profiling is enabled. The Enable/Disable Profiling icon  must be toggled on to enable profiling. Profiling may not be enabled if the program has started running or if it was explicitly disabled. Data will not be collected if profiling is not enabled.
3. Click on the Control tab of the Profile Setup window.
4. Click on the `zlib.pjt` project in the Project View to open the project file list.



5. Double click on source file `zlib.c` to open.
6. Go to the main function, and highlight the first opening brace (“{”) of the main function (line 711), and drag-and-drop it to the Halt Collection text line within the Control tab. This will halt the data collection in the beginning of the application. Your line number may vary.
7. In the source window, scroll up to locate the function `do_compress`.

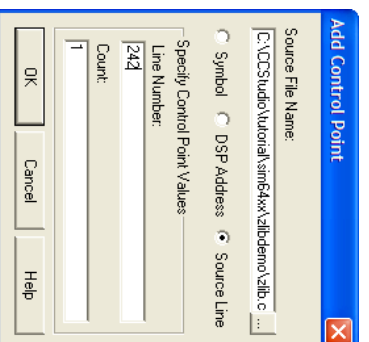
file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007

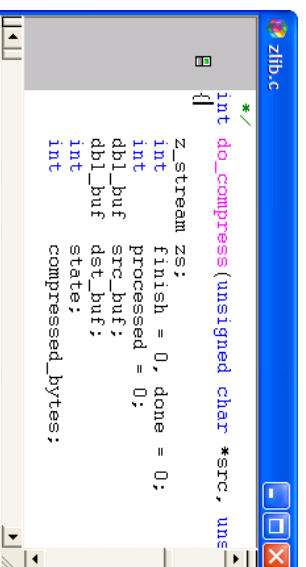
Getting Started With the Code Composer Studio Tutorial

Page 198 sur 548

8. Position your cursor at the opening brace (“{”) of the `do_compress` function (line 242). Your line number may vary.
9. Right-click on the Halt/ Resume Collection pane (lower pane) of the Profiler Setup window. Select Create Resume Collection point from the pop up menu.
10. Select the Source Line radio button, and type in the line number of the function (line 242). Your line number may vary. Select OK to add the Resume Collection point. You can also drag-and-drop the function from the `c` file to the Resume Collection Point text line within the Control tab.



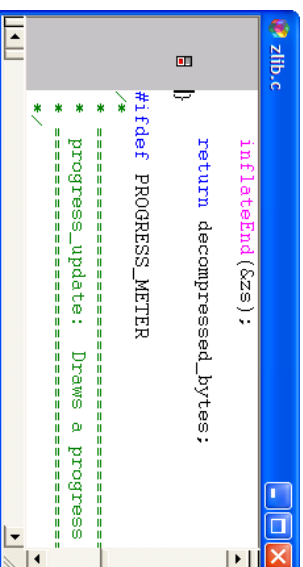
A green symbol appears, indicating the location of the Resume Collection point. If you see an ellipsis (...) instead, widen the selection margin until the symbol becomes visible.



file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007

11. Scroll down to locate the end of the function do decompress.
  12. Position your cursor at the closing brace ("}") of the function. Note the line number 553. Your number may vary.
  13. Right-click on the Halt/ Resume Collection pane (lower pane) of Profiler Setup window. Select Create Halt Collection point from the pop up menu.
  14. Select the Source Line radio button, and type in the line number of the function. Your line number may vary. Select OK to add the Halt Collection point
- A red symbol appears, indicating the location of the Halt Collection point.



```

zlib.c
-----
}
    inflateEnd(&zs);
    return decompressed_bytes;
}

#ifdef PROGRESS_METER
/*
 * -----
 * progress_update: Draws a progress
 * -----
 */

```

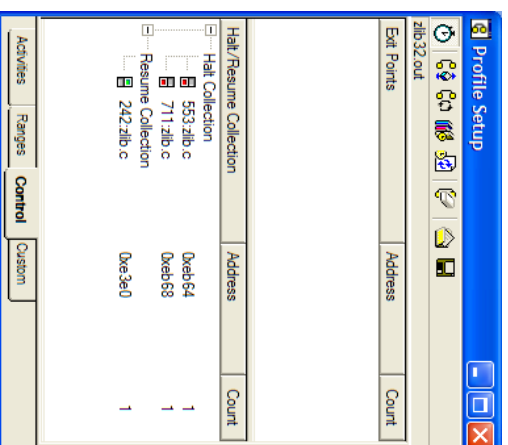
The added Halt Collection and Resume Collection points are also displayed in the Halt/Resume Collection Pane of Profile Setup window.

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007

Getting Started With the Code Composer Studio Tutorial

Page 200 sur 548



Halt/Resume Collection			
	Address	Count	
Halt Collection			
<input type="checkbox"/>	553:zlib.c	0x6b64	1
<input type="checkbox"/>	711:zlib.c	0x6b68	1
Resume Collection			
<input type="checkbox"/>	242:zlib.c	0x63e0	1

Because the application is compiled with optimization, you can only place Control points in certain locations. In general, the beginning or end of a function is the safest place for a Control point. Also, unless loops are optimized away, it is valid to place Control points at the opening or closing brace of a loop. The Control point must be set on a line that contains executable code.

For more details on the Ranges and Control tabs, refer to the Profile Setup help.

In the next lesson, we will select the collection option sets we would like CST to use to profile the application.



### Selecting Collection Option Sets

CST - L2

To understand the behavior of your application and the effects of different compiler options, CodeSizeTune must compile and run your application under a number of different collection options sets.

By default, CodeSizeTune has five different collection options sets defined, ranging from optimize for maximum speed to optimize for minimum code size. By default, three collection options

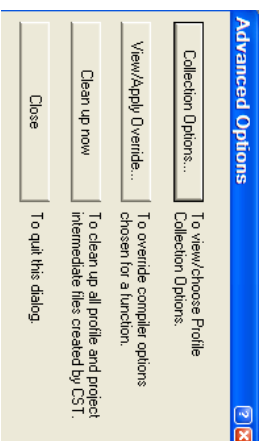
file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007

sets are activated. If a collection options set is activated, CodeSizeTune uses it when profiling your application. A deactivated collection options set is available, but is not used until activated.

To open the Choose Profile Collection Options dialog:

1. Click the Advanced Options (  ) button on the CodeSizeTune toolbar to open the Advanced Options dialog window.



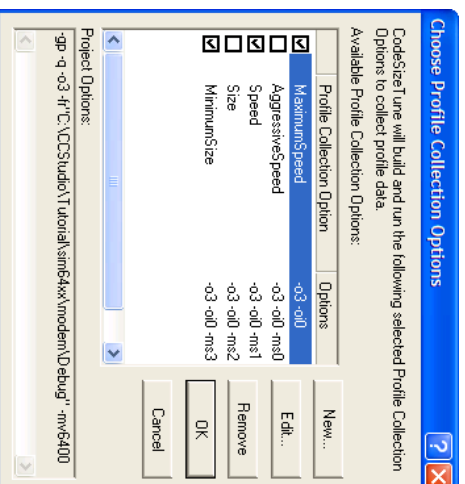
2. Select collection options from the dialog window. A checkbox with a checkmark indicates an activated collection options set:

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007

Getting Started With the Code Composer Studio Tutorial

Page 202 sur 548



While running the ZLIB example, you should use the default collection options set. When using CodeSizeTune with a real application, you may want to increase the number of activated collection options sets, or even define some of your own. The more collection options sets used, the longer CodeSizeTune takes to build and profile your application. This gives CodeSizeTune more options for an optimal build of your application, relevant to your speed or code size constraints.

3. Click OK to close the dialog box.

In the next lesson, we will use the collection options sets we have specified to build and profile the application.





### **Building and Profiling**

CST - L2

Now that we have prepared our application by defining a profile range, and set our collection options sets, we are ready to build and profile.

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007

1. To Build and Profile, click on Incremental Build and Profile (  ) in the CodeSizeTune toolbar.
2. This step can take some time. To monitor the progress of CodeSizeTune, click on the output window, then click on the CodeSizeTune tab. This will show you the build progress, first building, resetting the CPU, then loading, and finally profiling for each of the profiling collection options sets.
3. When CodeSizeTune finishes profiling your application under each collection options set, it displays the CodeSizeTune output graph.
4. To force CodeSizeTune to rebuild and profile collection options sets, select Rebuild All and ReProfile All (  ) from the CodeSizeTune toolbar. Otherwise, CodeSizeTune will not typically build and profile a collection option set if it has not changed since the last time it was used.

In the next lesson, you will view and select the different function option sets on the output graph you created here.



### Viewing and Selecting Function-Specific Option Sets

GST - L2

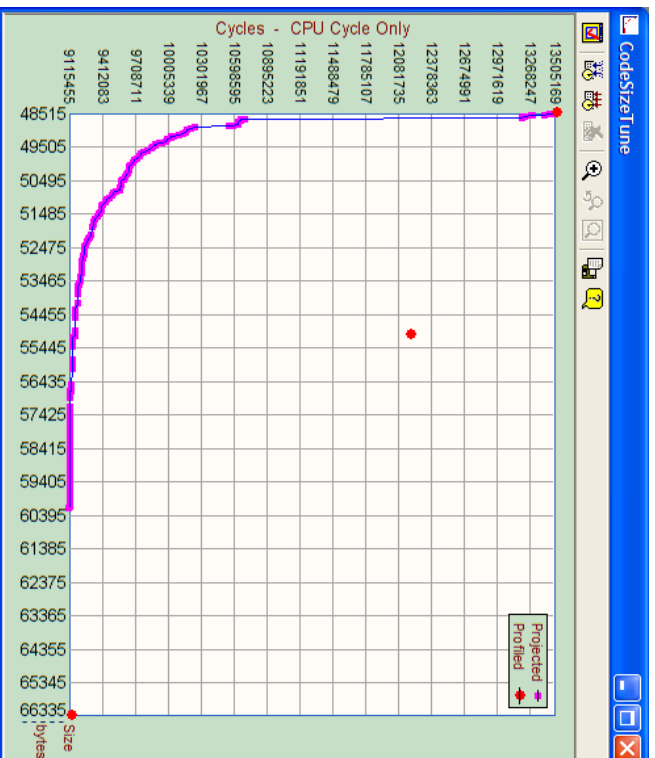
Once CodeSizeTune has built and profiled all active collection options sets, it graphs the set of all function-specific options sets in purple in the CodeSizeTune output window. There are profile collection options sets in red on the graph. The graph may look different depending on your resolution.

file:///C:/Documents and Settings/Pierre-Armand/Local Settings/Temp/~hhBA53.htm


26/09/2007

Getting Started With the Code Composer Studio Tutorial

Page 204 sur 548





**Note:** If the CodeSizeTune output window graph does not display after you have profiled, check to see if another window has hidden the output window inside the Code Composer Studio IDE.

1. Hold the mouse pointer over any of the points to display its code size and cycle-count information. Let's take a closer look at the graph and choose a function-specific options set.
2. Select Zoom (  ) from the toolbar.
3. Left-click to drag a border around the area of the curve you would like to examine more closely.
4. Continue this zooming process until you are satisfied with the level of detail.

file:///C:/Documents and Settings/Pierre-Armand/Local Settings/Temp/~hhBA53.htm

26/09/2007

If you would like to go back to the last level of zoom, click on the Undo-Zoom  button.

5. Deselect Zoom () to return your cursor to its normal appearance.
6. Pick one of the points (in purple) with the desired cycle and code size, and click on it to make it the Selected Function-Specific Options Set. The Profile Viewer window automatically displays when you select a point from the graph.

The Profile Viewer window displays detailed information about the selected function-specific options set. This information includes the size, cycle, name and options for each function. The comprehensive tab includes all profile data information. To see specific information for CodeSizeTune, select the CodeSizeTune tab.

In the next lesson, we will open the overrides window.



## View Overrides

CST - L2

In some cases, you may want to directly determine which options set is assigned to a particular function, rather than let CodeSizeTune 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. You can override the profiling collection option set that CodeSizeTune assigns to a function using the override window. The override is then applied to all of the options sets that CodeSizeTune computes.

To open the overrides dialog:

1. Click on the Advanced Options () button from CodeSizeTune toolbar.
2. From the dialog box, select View/Apply Overrides.

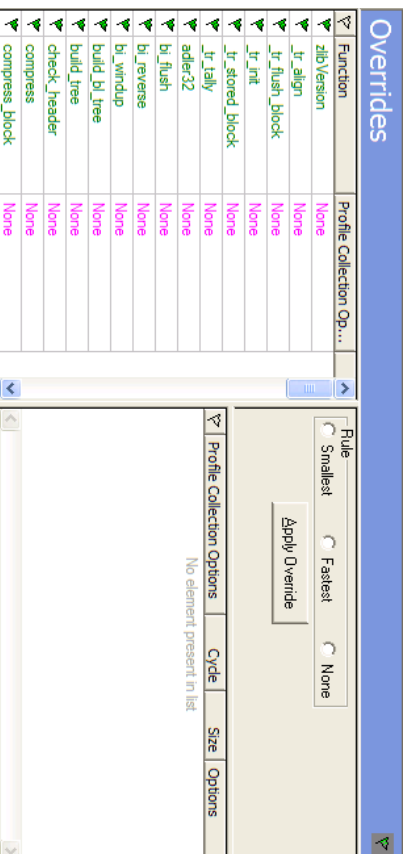
The Overrides window displays the overrides for the current function-specific option set. The Function column lists all functions. Any overrides applied to a function are indicated by a red flag next to the function name and listed in the Profile Collection Option/Rule column.

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007

Getting Started With the Code Composer Studio Tutorial

Page 206 sur 548



The details of the selected function are shown in the right-hand pane of the Overrides window.

The next lesson show you how to override the option sets that CodeSizeTune assigns to a function.



## Selecting Rules and Overrides

CST - L2

The right hand pane of the Overrides window provides two ways to override a function:

- The top-right half of this pane allows you to enter an override rule: smallest or fastest. The default value, None, clears the rule. Selecting a rule instructs CodeSizeTune to always choose the options set that meets the criterion specified.

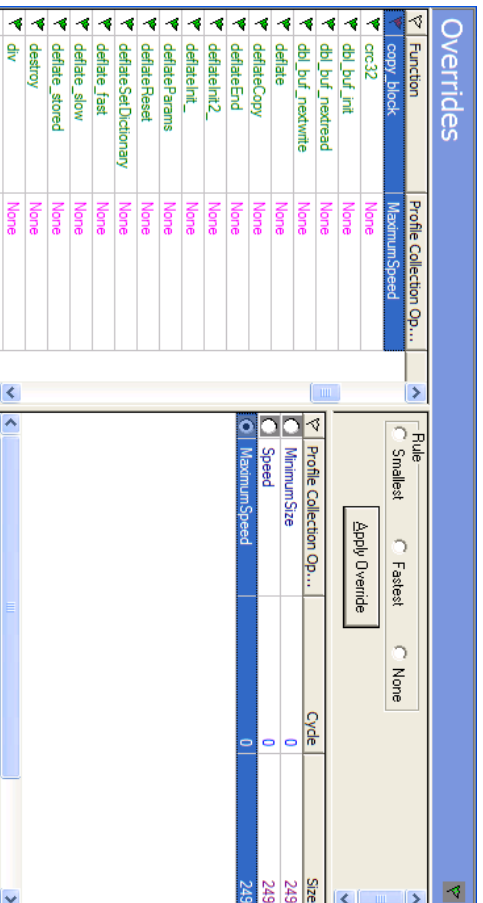
- The bottom-right half of this pane allows you to bind a particular options set to the function, so that particular options set is always assigned in all of the graphed points. This method is useful if the function is not invoked during the profiling session and CodeSizeTune has no run time data from which to choose the best option set. The None option can also be used to clear the selected Profile Option.

Assume that copy\_block is a time-critical function. You can force CodeSizeTune to always choose the maximum speed configuration for copy\_block by clicking the radio button next to the Maximum Speed collection option set.

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007

1. In the Overrides window, scroll down the function column and select copy\_block.
2. Click the radio button next to the Maximum Speed collection option set on the lower-right side.



3. To apply overrides to the CodeSizeTune graph, click Apply Overrides in the bottom of the Rule Method section of the Overrides window.
4. Click Yes twice to save the override changes, if the dialog requests it.

Once you click Apply Override, all function-specific options sets will now have copy\_block compiled with the Maximum Speed option, and it creates a new CodeSizeTune graph with your specific changes. If you rebuild the project, the overridden column of the Profile Viewer window will indicate "Yes" if the function has been overridden and "No" if the function has not.

In the next lesson, you will save the selected function-specific options set.



### Saving a Configuration

CST - L2


file:///C:/Documents and Settings/Pierre-Armand/Local Settings/Temp/~hhBA53.htm

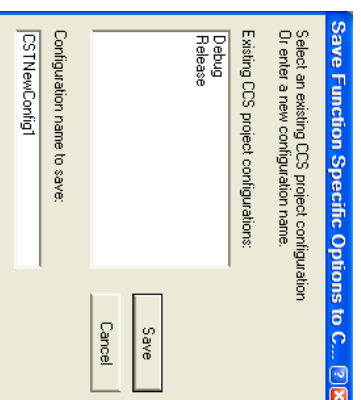
26/09/2007

### Getting Started With the Code Composer Studio Tutorial

Page 208 sur 548

The last step you need to perform is saving the selected function-specific option sets. Saving your selected settings adds function-level options defined by CodeSizeTune to a user-created project configuration.

1. Pick one of the points (in purple) with the desired cycle and code size and click on it to make it the Selected Function-Specific Options Set.
2. Click the Save  button from the CodeSizeTune toolbar.
3. Type the name of the new configuration, and click OK. The default name is CSTNewConfig1. Code Composer Studio will automatically use this just saved configuration.



**Note:** The -@filename option is added to the project options in the configuration you just created. This option tells the compiler how to use the function level options defined by CodeSizeTune.

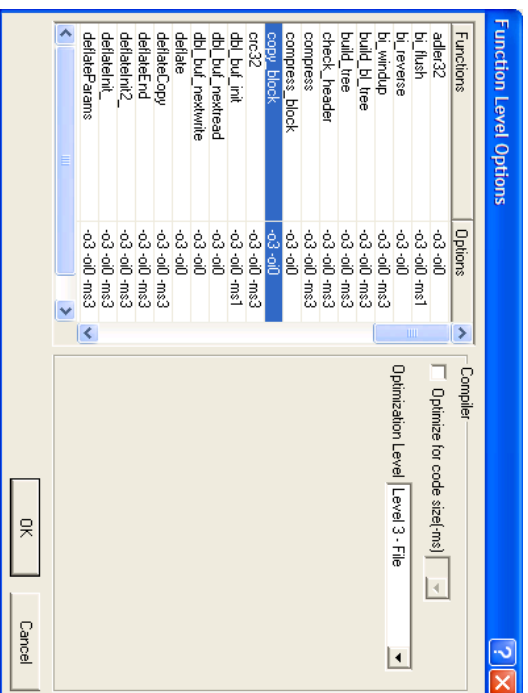
Next, let's look at the function level options CodeSizeTune added to your project.

4. From the main Project Configuration drop-down menu, select the configuration you just created to make it the active configuration.
5. From the Project menu, choose Function Level Options. In this dialog you can select a function and continue to change function level optimizations.

file:///C:/Documents and Settings/Pierre-Armand/Local Settings/Temp/~hhBA53.htm

26/09/2007





6. Click OK to close dialog box.

7. From the Project menu, choose Rebuild All. This action rebuilds the project using the new project configuration, which contains the function level build options defined by CodeSizeTune. The .out file is built into the *C:\CCStudio\_v3.10\tutorial\sim64xx\zlib\yourname* directory.

After saving your selected settings to a Code Composer Studio™ project, you can now continue to build your application, knowing that the optimal function-level set of compiler options are being used for your given speed and size constraints. If you would like to revert to a pre-CodeSizeTune configuration, select one of the other configurations from the Project Configuration drop-down menu.

This concludes the CodeSizeTune In-Depth tutorial.



## Compiler Consultant Tool Introduction

CCT - Intro

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007

## Getting Started With the Code Composer Studio Tutorial

Page 210 sur 548

In this tutorial, you will analyze a simple loop for optimization. The Compiler Consultant Tool analyzes your application and makes recommendations for changes to optimize the performance. The tool displays two types of information for the user: Compile Time Loop Information and Run Time Loop Information. Compile Time Loop Information is created by the compiler. Each time you compile or build your code, Consultant will analyze the code and create suggestions for different optimization techniques to improve code efficiency. These suggestions include compiler optimization switches and pragmas to insert into your code that give the compiler more application information. You then have the option of implementing the advice and building the project again. The Run Time Loop Information is data gathered by profiling your application. This run time loop information helps prioritize the advice.

Learning Objectives:

- Use the Compiler Consultant Tool to create optimization advice on a sample project loop
- Introduce the different kinds of advice used to optimize the project, including Options, Alias, Alignment\* and Trip Count\* Advice
- Implement the advice, and observe the effects

### Examples used in this lesson: consultant

Target Configuration: This tutorial should be executed either on the C6416 Device Cycle Accurate Simulator, Little Endian, or the C6211 Device Cycle Accurate Simulator, Little Endian. Using a different version of the compiler may cause cycle counts to change slightly, or different advice to be generated. Executing the tutorial on different execution platforms may require modification of the linker command file due to differences in the memory map. For more information on the Setup utility, access the [Configuring Target Devices](#) tutorial or the online help.

### Application Objective:

This tutorial uses the consultant project to display different features of the Compiler Consultant Tool. You will analyze the example loop using the Compiler Consultant tool throughout the optimization process. After each optimization, new advice will appear based on the changes made. Each type of advice has specific steps associated with it. You will implement these steps and view the results. Optimization will continue until the Consultant Compiler Tool has determined that the loop has been fully optimized.

This tutorial consists of the following steps. To go directly to a lesson, click on the link below:

If you are using a 6211 target, click [here](#).

If you are using a 6416 target, click [here](#).

\*In this tutorial, Alignment and Trip Count advice appear with C64xx devices only.

## Getting Started with Consultant

CCT - L1

In this lesson, you will analyze a project called consultant.plt. You will use the Profile Viewer to analyze the project, and the Compiler Consultant Tool will present advice on how to optimize the loop.

Over the course of the lessons, you will utilize the following types of files:

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007

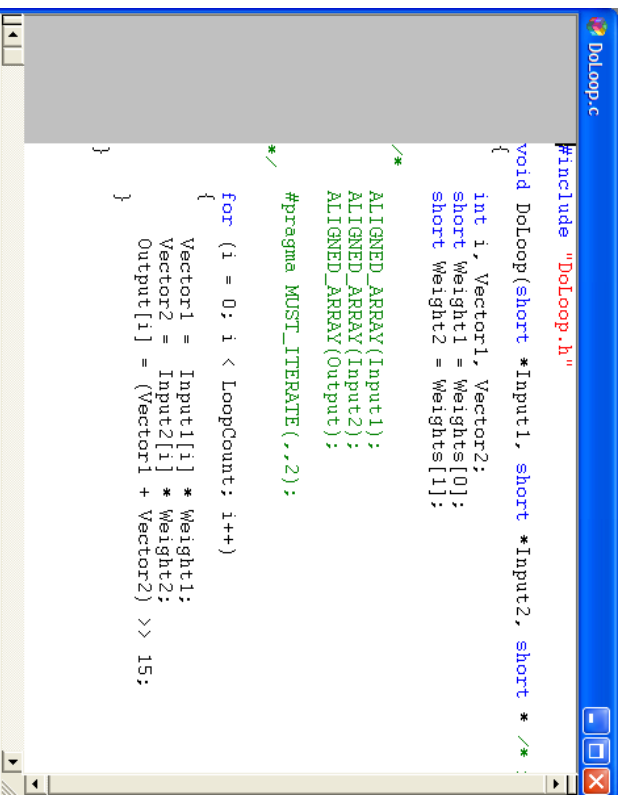
- .c This file contains source code that provides the main functionality of this project
  - .h This file contains the function prototypes
  - .jdt This file contains all of your project build and configuration options
1. If you have not already done so, from the Windows Start menu, choose Programs → Texas Instruments → Code Composer Studio 3.1 → Code Composer Studio 3.1. (Or, double-click the Code Composer Studio icon on your desktop.)
  2. From the Project menu, choose Open, and browse to: C:\CCStudio\_v3.10\tutorial\sim64xxx\consultant.
  3. Select consultant.pjt, and click Open to load the project.
  4. Click on the + sign next to the consultant.pjt project in the Project View Window to open the project file list.
  5. Click on the + sign next to the Source folder inside consultant.pjt. Navigate to the Doloop.c and double-click to open it.

file:///C:/Documents and Settings/Pierre-Armand/Local Settings/Temp/~hhBA53.htm

26/09/2007

Getting Started With the Code Composer Studio Tutorial

Page 212 sur 548



```
#include "Doloop.h"

void Doloop(short *Input1, short *Input2, short *
{
    int i, Vector1, Vector2;
    short Weight1 = Weights[0];
    short Weight2 = Weights[1];

    /*
    ALIGNED_ARRAY(Input1);
    ALIGNED_ARRAY(Input2);
    ALIGNED_ARRAY(Output);
    */
    #pragma MUST_ITERATE(,2);

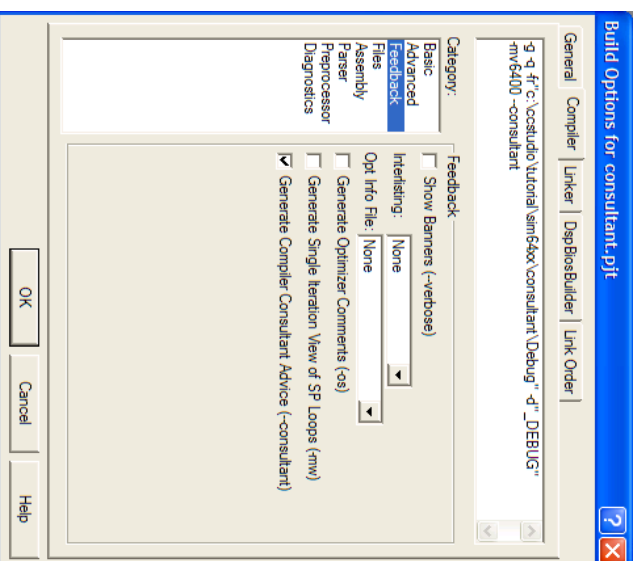
    for (i = 0; i < LoopCount; i++)
    {
        Vector1 = Input1[i] * Weight1;
        Vector2 = Input2[i] * Weight2;
        Output[i] = (Vector1 + Vector2) >> 15;
    }
}
```

Doloop() is called directly from main() one time. It contains only one loop. This loop is the focus of the tutorial.

6. Under the Project menu, choose Build Options...
7. In the Build Options dialog, click the Compiler tab.
8. Click on the Feedback item in the Category list.
9. Click on the checkbox Generate Compiler Consultant Advice (--consultant).
10. Your build options should resemble the following, although the path may change depending on your target:

file:///C:/Documents and Settings/Pierre-Armand/Local Settings/Temp/~hhBA53.htm

26/09/2007



11. Click OK to close the Build Options dialog.
12. From the Project menu, choose Rebuild All.
13. From the Profile menu, choose Viewer.
14. Once the Profile Viewer window appears, click on the Consultant tab. The one line in the Profile Viewer represents the single loop in the Consultant tutorial.

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007

Getting Started With the Code Composer Studio Tutorial

Page 214 sur 548

Address Range	Symbol Name	SLR	cycle.CPUExcl. ...	Estimated Cycle...	Estimated Start...	Advice Count	Advice List
Loop #1	Doloop	17-22.Doloop.c	64	0	2	(2) Options	

Notice that the [Estimated Cycles Per Iteration](#) for the loop is currently 64 cycles, and the [Estimated Start Cycles](#) is zero. Also note that the loop has two pieces of Options advice, as indicated by the Advice Count column in the Profile Viewer. For information on the columns in Profile Viewer, consult the online help file: .

The default row height in the profile viewer only allows you to see one advice type in the Advice List column. To increase row height, place the mouse pointer on the solid horizontal line at the bottom of the top row in the Address Range column. Double-click on that line, and the row height will automatically resize to fit all the information.

In the next lesson, you will open the Advice window and analyze the Compiler Consultant advice.



### Options Advice Implementation

CCT - L1

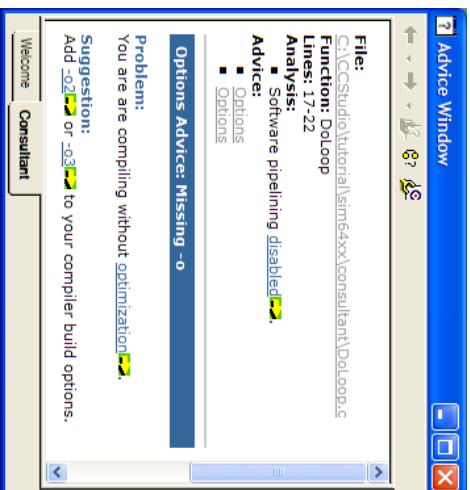
In this lesson, you will analyze the options advice given by the Consultant Tool, and implement it for the consultant.pjt project.

1. Double click on the Advice List cell for the Doloop row.

The Advice Window appears, and the Consultant tab displays advice for the Doloop function. The Analysis of the Doloop functions indicates that the software pipelining optimizations in the compiler are currently disabled. Also note the two pieces of Options advice.

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007



2. Click on the link to the first piece of Options advice.

The first piece of Options advice states we are compiling without optimization. The word optimization appears underlined and in color, similar to a link in a typical HTML document. In Code Composer Studio, compiler terminology in advice is often linked to the online help, as indicated by the online help icon.

The suggested solution indicates you should turn on the `-o2` or `-o3` compiler option. Before doing so, let's take a quick look at the second piece of advice.

3. Scroll back to the top of the Advice window with the scroll bar.

4. Click on the link to the second piece of Options advice in the advice list.

The second piece of Options advice states that we are compiling with the `-g` debug option turned on. As the advice notes, compiling with `-g` hurts performance. The advice suggests turning off the `-g` option.

5. Under the Project menu, choose Build Options...
6. In the Build Options dialog, click the Compiler tab.
7. Click on the Basic item in the Category list.

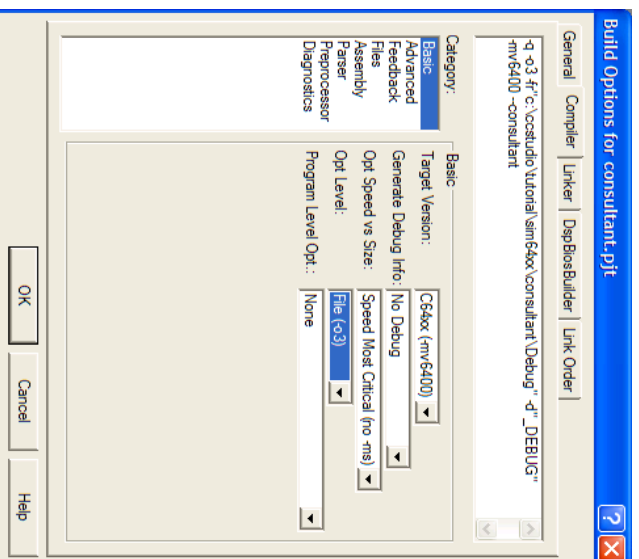
file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007

Getting Started With the Code Composer Studio Tutorial

Page 216 sur 548

8. From the Generate Debug Info drop down list, choose No Debug.
9. From the Opt Level drop down list, choose File (`-o3`).
10. Your build options should be similar to:



11. Click OK to close the Build Options dialog.
12. From the Project menu, choose Build.
13. After the build, the advice in the Advice Window is no longer valid, so it clears the advice and informs the user that double clicking on the Advice Count column will refresh the advice.

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007

14. In the Profile Viewer, click on the Consultant tab. If it is not active, and double-click on the Advice Count cell for the Loop with an Estimated Cycles Per Iteration of 11 to update the Advice Window for this loop. You can also increase the row heights for the two loops in the Profile Viewer again to see all the pieces of advice.

Address Range	Symbol Name	SIR	cycle/CPU/Excl. ...	Estimated Cycle	Estimated Start...	Advice Count	Advice List
Loop #1	D0Loop	17-22:D0Loop.c		11	0	5	(2) Alias (3) Alignment
Loop #2	D0Loop	17-22:D0Loop.c		2	9	6	(2) Alias (3) Alignment (1) Trip count

There are now five pieces of advice for the main loop in the profile viewer, two Alias and three Alignment. Before looking at the specifics on these advice topics, the next lesson will cover loop analysis.



### Loop Analysis

CCT - LI

Look at the Consultant tab of the Advice Window. The Analysis for this loop indicates it has been duplicated. Loop duplication can occur when the compiler is unable to determine if one or more pointers are pointing at the same memory address as some other variable. Such pointers are called **aliases**. One version of the loop presumes the presence of aliases, the other version of the loop does not. The compiler generates code that checks, at runtime, whether certain aliases are present. Then, based on that check, the compiler executes the appropriate version of the loop. The copy of the loop presently displayed in the Advice Window executes when the runtime check for aliases indicates that aliases may be present.

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007

Getting Started With the Code Composer Studio Tutorial

Page 218 sur 548

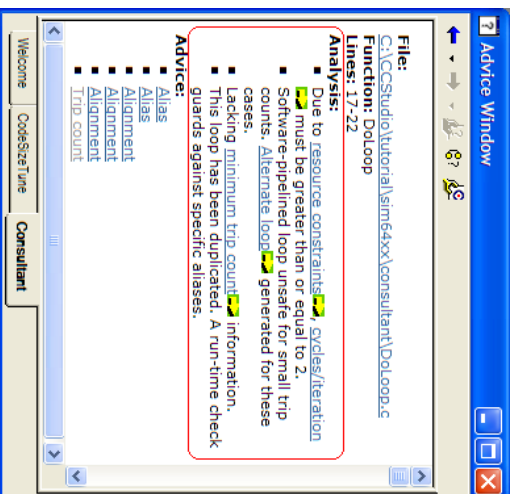
1. Double-click on the Advice Count cell in Profile Viewer for the other row.

Look at the analysis of that loop. It has a similar note about a duplicated loop, but it presumes certain aliases are not present.

Further note the bullet about the alternate loop. This is a loop that is executed when a different runtime check for a minimum number of loop iterations has failed. This alternate loop is not software pipelined. For this reason, the alternate loop has no entry in the Profile Viewer.

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007



This pseudo-code summarizes the relationship between the three variations of the loop.

```

if (enough loop iterations)
{
    if (aliases are present)
        loop #1
    else
        loop #2
}
else
    alternate loop // not in profile viewer

```

We can profile the code to see which loop versions get selected when the code is executed.


- From the File menu, choose Load Program to start the program load.
- Browse to [C:\CCStudio\\_v3.10\tutorial\sim64xx\consultant\Debug](#).
- Choose consultant.out.

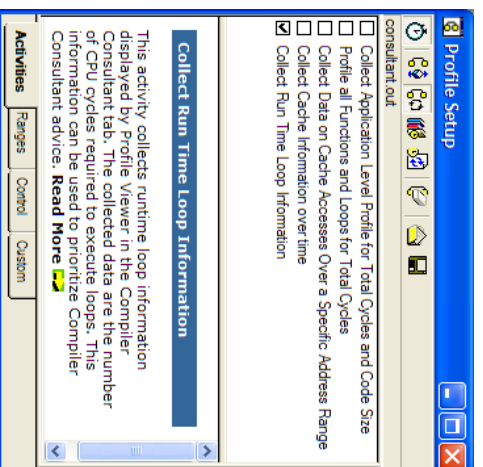
file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007

Getting Started With the Code Composer Studio Tutorial

Page 220 sur 548

- Click Open.
- From the Profile menu, choose Setup.
- Select the Activities tab near the bottom of the window.
- Check the icons at the top of the Profile Setup tool to determine if profiling is enabled. The Enable/Disable Profiling icon  must be toggled on to enable profiling. Profiling may not be enabled if the program has started running or if it was explicitly disabled. Data will not be collected if profiling is not enabled.
- Place a check next to Collect Run Time Loop Information.



This activity will count the cumulative total of CPU cycles, ignoring system effects, executed in each version of the loop. System effects that it ignores include things like cache misses, memory bank conflicts, off-chip memory access latency, etc.

**Note:** Your execution platform may not support collection of the event cycle CPU, and so will not show the activity Collect Run Time Loop Information. In that case, choose the activity Profile All Functions and Loops for Cycles instead. This activity also counts the cumulative total of the CPU cycles, but includes system affects. Further, in addition to profiling all versions of the loop, it profiles each function. In the sections which follow, use the column cycle.Total:Excl. Total in place of cycle.CPU:Excl. Total. For more information on runtime profiling events access, see the online help file: .

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007

10. Run the program by choosing Debug →Run.
11. Wait until the program halts. One indicator is the message Halted in the lower left corner.
12. In the profile viewer, click on the Consultant tab if it is not active, and hover the mouse over the column that starts with cycle:CPU:Excl. to see that the full name of the column is "cycle:CPU:Excl. Total".

Address Range	Symbol Name	S/R	cycle:CPU:Excl. ...	Estimated Cycle...	Estimated Start...	Advice Count	Advice List
0:0x0c-0xd4	DdlLoop	17-22:DdlLoop.c	680	11	0	5	(2) Alias...
0:0x140-0x1e4	DdlLoop	17-22:DdlLoop.c	0	2	9	6	(2) Alias...

Note the loop with the 6 pieces of advice has 0 in the cycle:CPU:Excl. Total column. This means the loop for which the analysis states aliases are not present did not execute. The loop that did execute is the one for which aliases are presumed to be present. That is because the runtime check for aliases concluded that aliases may be present. In fact, there are no aliases among these pointers. The runtime check concluded that aliases may be present because it is too conservative; it was too cautious.

To clarify this point, please find the pseudo-code below with comments (in green) to indicate the actual path of execution.

```

if (enough loop iterations)
{
    /* executes because there are enough loop iterations */
    if (aliases are present)
        /* executes because test for aliases is too conservative */
        loop #1
    else
        /* does not execute */
        loop #2
}
else
    /* does not execute */
    alternate loop // not in profile viewer

```

The loops are presently ordered, from highest to lowest, by the number of cycles executed. This is unlikely to happen in a real application. Typically, information from many loops is present, and sorting the loops is a good way to determine which to focus on first. You can sort on any column by double-clicking on the column header.

13. Double click multiple times to sort the rows, toggling between ascending and descending order.  
We will now focus on the executed loop that currently takes 680 cycles. As we implement the alias and trip count advice for this loop, the compiler will no longer generate the other versions of the loop, and they will no longer appear in the Profile Viewer.

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007

Getting Started With the Code Composer Studio Tutorial

Page 222 sur 548

14. Double click on the Advice Count cell for the loop which takes 680 cycles.

There are five pieces of advice for the main loop, two Alias and three Alignment. We will first look at the Alias advice and solve it, before moving on to the Alignment advice.



### Alias Advice Implementation

CCT - L1

In this lesson, you will analyze the alias advice given by the Compiler Consultant Tool, and implement it for the consultant.gjt project.

1. In the Consultant tab of the Advice Window, click on the link to the first piece of Alias advice.

The problem statement indicates the compiler cannot determine if two pointers may point to the same memory location, and therefore cannot apply more aggressive optimizations to the loop. The Consultant Tool offers several suggestions to solve the problem.

**Problem:**  
The compiler cannot determine whether **Input1** (C:\CCStudio\Tutorial\sim64xx\consultant\DdlLoop.c:3) might access the same memory locations as **Output** (C:\CCStudio\Tutorial\sim64xx\consultant\DdlLoop.c:3) within the loop.

**Suggestion:**  
Restrict qualify the definition of **Input1**, if the safety **CRITERIA** can be met.

**Example:**

**1st Alternate Suggestion:**  
Restrict qualify the definition of **Output**, if the safety **CRITERIA** can be met.

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007

The primary suggestion tells us to use the restrict qualifier on the Input1 pointer if the *safety criteria* can be met. Inspect the call to the function Doloop, and notice that the array passed in as the argument for Input1 is the global array also named Input1. This array does not overlap any of the arrays passed in for the other arguments. This means that the memory referenced by the Input1 variable cannot be referenced by any other variable in the function. Therefore, the Input1 variable meets the safety criteria required for correct use of the restrict keyword. Before we apply the primary suggestion, let's take a look at the alternate suggestions.

The first alternate suggestion indicates that using the restrict qualifier on the Output pointer could solve the problem. Finally, the second and third alternate suggestions indicate that restrict qualifying a local copy of the Output or Input1 pointer would also solve the problem.

2. Go back the top of the advice window and click the link to the second piece of Alias advice.

This piece of alias advice is similar to the first, except it mentions pointers Output and Input2.

Because there are two pieces of alias advice, one for Output and Input1, and the other for Output and Input2, restrict qualifying Output resolves both problems.

3. Click on the link that says Output in the Problem section of the Advice Window to open the Doloop.c file on the line that declares Output as an argument passed in to the Doloop function.

4. Modify the Doloop.c to remove the C comment delimiters `/**/` around the restrict qualifier to the Output pointer parameter.

Before:

```
void Doloop(short *Input1, short *Input2, short /* restrict*/ Output, short *Weights, int LoopCount)
```

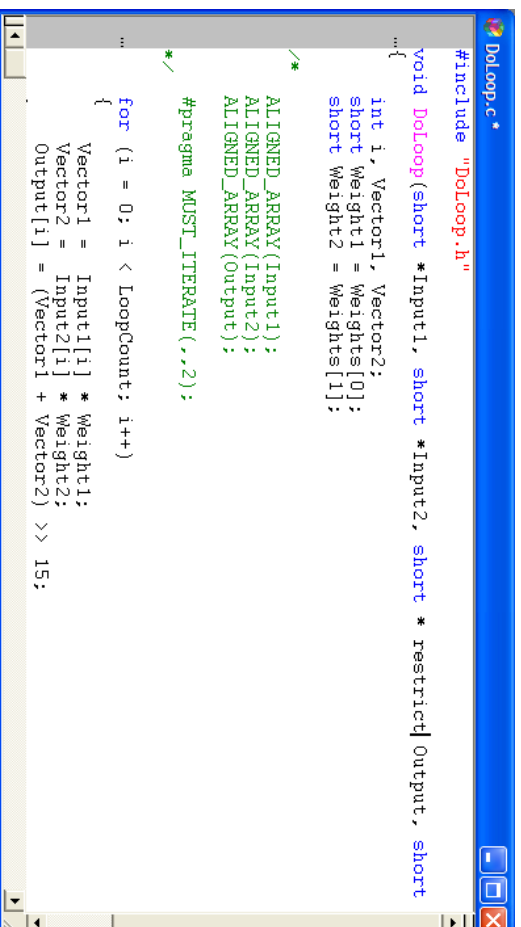
After:

file:///C:/Documents and Settings/Pierre-Armand/Local Settings/Temp/~hhBA53.htm

26/09/2007

Getting Started With the Code Composer Studio Tutorial

Page 224 sur 548



```
Doloop.c *
#include "Doloop.h"

...{
void Doloop(short *Input1, short *Input2, short * restrict Output, short
    int i, Vector1, Vector2;
    short Weight1 = Weights[0];
    short Weight2 = Weights[1];
}
/*
  ALIGNED_ARRAY(Input1);
  ALIGNED_ARRAY(Input2);
  ALIGNED_ARRAY(Output);
*/
#pragma MUST_ITERATE(,,2);
...
for (i = 0; i < LoopCount; i++)
{
    Vector1 = Input1[i] * Weight1;
    Vector2 = Input2[i] * Weight2;
    Output[i] = (Vector1 + Vector2) >> 15;
}
```

5. Save the file with menu item File→Save, or Control+S.

6. In the Project View, open the Include folder and double-click on the Doloop.h file.

7. Remove the C comment delimiters `/**/` around the restrict qualifier to the Output pointer parameter.

Before:

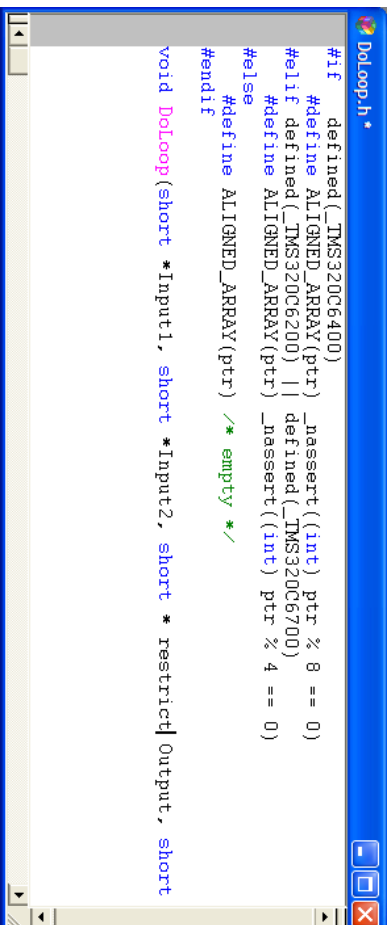
```
void Doloop(short *Input1, short *Input2, short /* restrict*/ Output, short *Weights, int LoopCount);
```

After:

file:///C:/Documents and Settings/Pierre-Armand/Local Settings/Temp/~hhBA53.htm

26/09/2007





```

Doloop.h
#If defined (_TMS320C6400)
#define ALIGNED_ARRAY(ptr) _nassert((int) ptr % 8 == 0)
#else
#define ALIGNED_ARRAY(ptr) _defined(_TMS320C6700)
#endif
#define ALIGNED_ARRAY(ptr) /* empty */
#endif


void Doloop(short *Input1, short *Input2, short * restrict Output, short

```

8. Save the file with menu item File→Save, or Control+S.

Source changes are complete, so it is time to build. However, building the project removes the cycle count Profile Viewer data, because it will be invalid. The following steps show how to save Profile Viewer data.

9. In the Profile Viewer toolbar on the left, click the icon  for Save Current Data Set.
10. Browse to C:\CCStudio\_v3.10\tutorial\sim64xx\consultant\Debug.
11. Enter run1 for the file name.
12. Select Dataset (\*\*.xml) for the Save As type, and click Save.

To see this data again later, you can use the Load Data Set  button in the Profile Viewer.

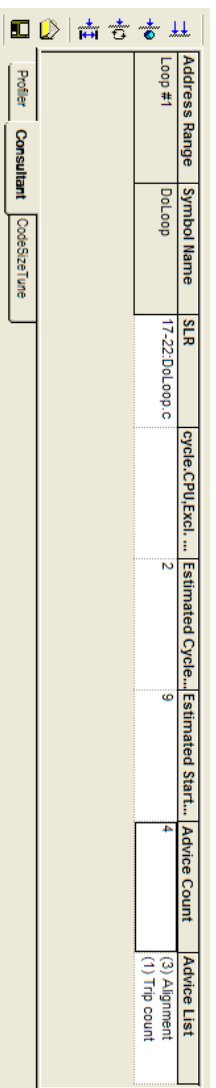
13. From the Project menu, choose Build.
- Note the cycle count data remains in the Profiler tab of the Profile Viewer even after a build, and remains there until another round of profile data is collected.
14. In the Profile Viewer, click on the Consultant tab if it is not active, and double-click on the Advice Count call for Doloop to update the Advice Window for this loop. You can also increase the row height for the loop in the Profile Viewer again to see more information.

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007

Getting Started With the Code Composer Studio Tutorial

Page 226 sur 548



Loop #1	Address Range	Symbol Name	SLR	cycle.CPUExcl. ...	Estimated Cycle...	Estimated Start...	Advice Count	Advice List
		Doloop	17-22 Doloop.c		2	9	4	(3) Alignment (1) Trip count

Notice that the Estimated Cycles Per Iteration for the loop has been reduced from 11 cycles to 2 cycles, while the Estimated Start Cycles has changed from zero cycles to 9 cycles.

There are now four pieces of advice, three Alignment and one Trip Count. We will look at the Alignment advice first and solve it before moving on to the Trip Count advice.



**Alignment and Trip Count Advice Implementation**

CCT - LI

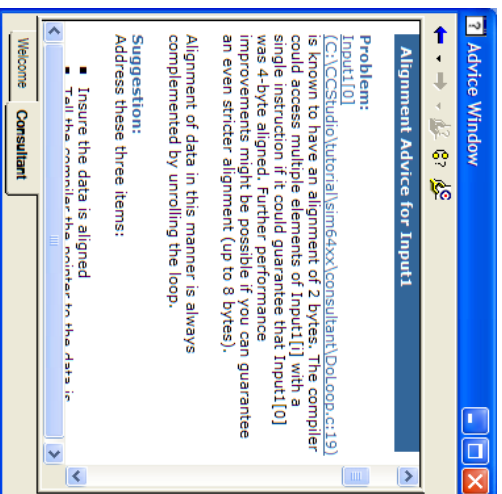
In this lesson, you will analyze the alignment and trip count advice given by the Consultant Tool, and implement it for the consultant.pjt project.

### Alignment Advice

1. In the Consultant tab of the Advice Window, click on the link to the first piece of Alignment advice.

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007



The problem statement indicates the compiler could access multiple elements of the Input1 array if it could verify that the array were aligned on a four- or eight-byte boundary. The suggestion recommends addressing three items: aligning the data, indicating the pointers are aligned, and unrolling the loop.

- The help entry on aligning data notes that arrays are automatically aligned on an 8-byte boundary for C64xx, and a 4-byte boundary for C62xx and C67xx. The function Doloop receives the base address of an array for all three pointer arguments. So, no further steps are required to align the data.
- For indicating the pointer holds the base address of an array, the help entry suggests a macro. So, we have the macro in the header file Doloop.h, as it is a more comprehensive solution than the simple `_nassert` given in the advice.
- For unrolling the loop, the advice suggests the use of the `MUST_ITERATE` pragma. For this simple case, we could indicate the multiple of loop iterations is 40. However, 2 is a more realistic loop count multiple.

The purpose of the `MUST_ITERATE` pragma is to tell the compiler about properties of the trip count of the loop. The trip count is the number of times the loop iterates. The syntax is:

```
#pragma MUST_ITERATE(min, max, multiple);
```

The arguments `min` and `max` are programmer-guaranteed minimum and maximum trip counts. The trip count of the loop must be evenly divisible by `multiple`. All arguments are optional. For information on the `MUST_ITERATE` argument, see the online help file:

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007

Getting Started With the Code Composer Studio Tutorial

Page 228 sur 548

The next two pieces of Alignment advice indicate the advice for `Input1` is also applicable for the pointers `Input2` and `Output`. This simple tutorial only calls `Doloop` once. In a real application, every call to `Doloop` must be inspected to insure that the base of an array is always passed to the pointers, and the loop iteration count is always a multiple of 2.

2. If the source file is not already open, click on the source file name (C:\CCStudio\_v3.10\tutorial\sim64xx\consultant\Doloop.c) at the top of the Advice Window to bring up the `Doloop.c` file in the Editor.
3. `Doloop.c` already contains all of the indicated changes as comments. Remove the comment delimiters (`/* */`) from the `ALIGN_ARRAY` macro and the `MUST_ITERATE` pragma.

Code example before (red text indicates area of change)

After:

```
Doloop.c *
#include "Doloop.h"

...{
    void Doloop(short *Input1, short *Input2, short * restrict Output, short
        int i, Vector1, Vector2;
        short Weight1 = Weights[0];
        short Weight2 = Weights[1];

        ALIGN_ARRAY(Input1);
        ALIGN_ARRAY(Input2);
        ALIGN_ARRAY(Output);

        #pragma MUST_ITERATE(40,2);

        for (i = 0; i < LoopCount; i++)
        {
            Vector1 = Input1[i] * Weight1;
            Vector2 = Input2[i] * Weight2;
            Output[i] = (Vector1 + Vector2) >> 15;
        }
    }
}
```

4. Save the file with File→Save or control+S.
5. From the Project menu, choose Build.
6. In the Profile Viewer, click on the Consultant tab if it is not active, and double-click on the `Doloop Advice Count` column to update the Advice Window.

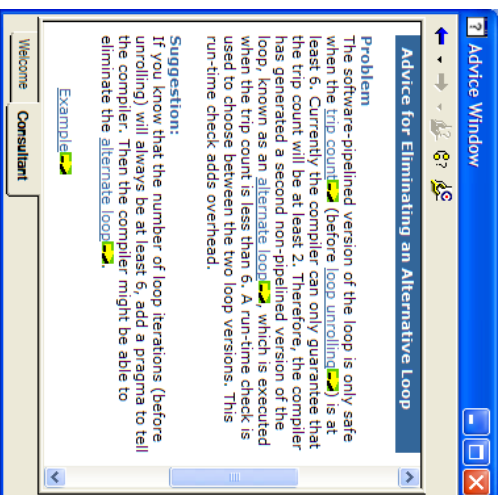
file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007

Notice that the Estimated Cycles Per Iteration for the loop improves to 1 cycle. The Estimated Start Cycles is now 13 cycles.

### Trip Count

1. In the Advice Window, click on the link to the Trip Count advice.



The problem statement indicates the compiler has no guarantee about the trip count of the loop. The primary suggestion suggests the use of the `MUST_ITERATE` pragma to indicate that the loop iterates at least 6 times. The alternate advice suggests using the `-m` compiler option. In our case, we know that the loop must iterate 40 times, so we will apply the `MUST_ITERATE` pragma. In a real application, every call to `Doloop` must be inspected to insure a loop count of at least 40.

2. If the source file is not already open, click on the source file name (`C:\CCStudio_v3.10\tutorial\sm64xx\consultant\Doloop.c`) at the top of the Advice Window to bring up the `Doloop.c` file in the Editor.
3. Add 40 as the first argument to the `MUST_ITERATE` pragma.  
Before:

```
#pragma MUST_ITERATE(,2);
```

After:

```
Doloop.c *
#include "Doloop.h"

...{
void Doloop(short *Input1, short *Input2, short * restrict Output, short
int i, Vector1, Vector2;
short Weight1 = Weights[0];
short Weight2 = Weights[1];

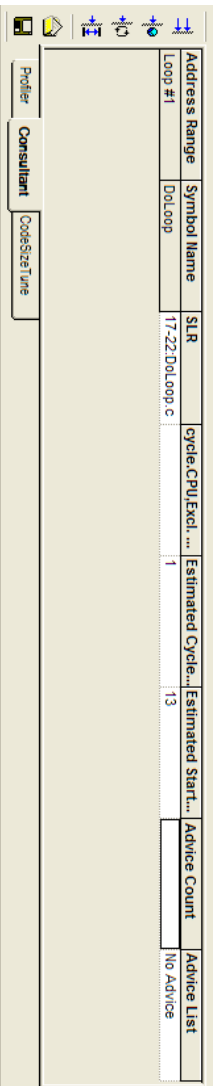
    ALIGNED_ARRAY(Input1);
    ALIGNED_ARRAY(Input2);
    ALIGNED_ARRAY(Output);
    #pragma MUST_ITERATE(40,2);

    for (i = 0; i < LoopCount; i++)
    {
        Vector1 = Input1[i] * Weight1;
        Vector2 = Input2[i] * Weight2;
        Output[i] = (Vector1 + Vector2) >> 15;
    }
}
```

4. Save the file with File->Save or control+S.
5. From the Project menu, choose Build.
6. In the Profile Viewer, click on the Consultant tab if it is not active, and double-click on the `Doloop Advice Count` column to update the Advice Window.

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm  
Getting Started With the Code Composer Studio Tutorial

26/09/2007  
Page 230 sur 548



Address Range	Symbol Name	SLR	cycle,CPU,Excl. ...	Estimated Cycle...	Estimated Start...	Advice Count	Advice List
Loop #1	DoLoop	17-22	DoLoop.c	1	13		No Advice

The Estimated Cycles Per Iteration remains at 1 cycle, and the Estimated Start Cycles remains at 13.

But the alternate loop, for low trip counts, is eliminated. This saves a small amount of runtime overhead, and fair amount of code space.

The Advice Count is now 0, and we have reduced the total cycle time from an initial value of  $64 * 40 = 2560$  cycles to  $13 + (1 * 40) = 53$  cycles, or roughly a 48X improvement.

Throughout this tutorial we have been applying compiler optimization changes that not only affected performance, but also affected the code size. Code size versus performance is a trade-off that only the designer can make. To help you evaluate code size versus performance for the DoLoop function, you can use the [CodeSizeTune tool](#).

For more information, see the application note *Introduction to Compiler Consultant*.



### Getting Started with Consultant

CCT - LI

In this lesson, you will analyze a project called consultant.pjt. You will use the Profile Viewer to analyze the project, and the Compiler Consultant Tool will present advice on how to optimize the loop.

Over the course of the lessons, you will utilize the following types of files:

- .c This file contains source code that provides the main functionality of this project
  - .h This file contains the function prototypes
  - .pjt This file contains all of your project build and configuration options
1. If you have not already done so, from the Windows Start menu, choose Programs → Texas Instruments → Code Composer Studio 3.1 → Code Composer Studio 3.1. (Or, double-click the Code Composer Studio icon on your desktop.)

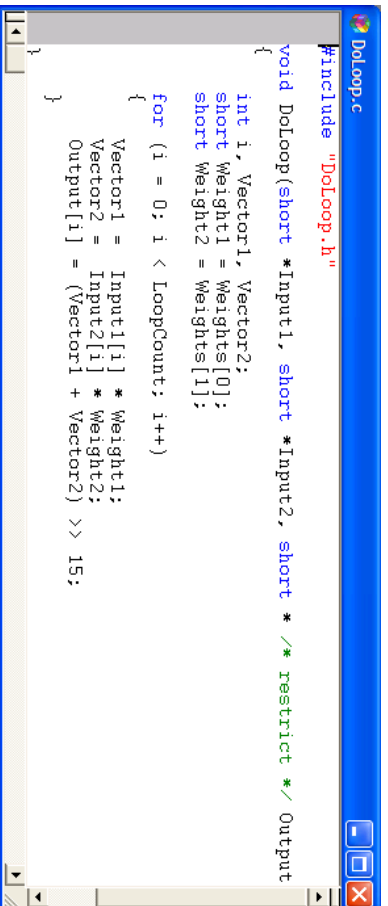
file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007

### Getting Started With the Code Composer Studio Tutorial

Page 232 sur 548

2. From the Project menu, choose Open, and browse to: C:\CCStudio\_V3.10\tutorial\sim62xx\consultant.
3. Select consultant.pjt, and click Open to load the project.
4. Click on the + sign next to the consultant.pjt project in the Project View Window to open the project file list.
5. Click on the + sign next to the Source folder inside consultant.pjt. Navigate to the DoLoop.c and double-click to open it.



```

Dolloop.c
#include "DoLoop.h"

void DoLoop(short *Input1, short *Input2, short * /* restrict */ Output
{
    int i, Vector1, Vector2;
    short Weight1 = Weights[0];
    short Weight2 = Weights[1];

    for (i = 0; i < LoopCount; i++)
    {
        Vector1 = Input1[i] * Weight1;
        Vector2 = Input2[i] * Weight2;
        Output[i] = (Vector1 + Vector2) >> 15;
    }
}

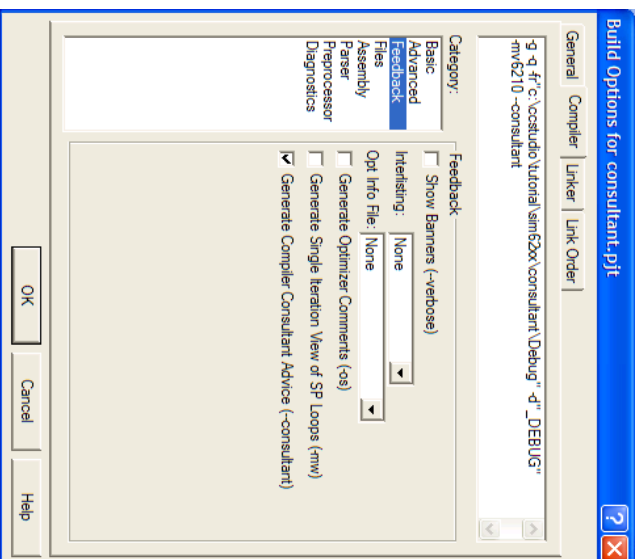
```

DoLoop() is called directly from main() one time. It contains only one loop. This loop is the focus of the tutorial.

6. Under the Project menu, choose Build Options...
7. In the Build Options dialog, click the Compiler tab.
8. Click on the Feedback item in the Category list.
9. Click on the checkbox Generate Compiler Consultant Advice (~-consultant).
10. Your build options should resemble the following, although the path may change depending on your target:

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007



11. Click OK to close the Build Options dialog.
12. From the Project menu, choose Rebuild All.
13. From the Profile menu, choose Viewer.
14. Once the Profile Viewer window appears, click on the Consultant tab. The one line in the Profile Viewer represents the single loop in the Consultant tutorial.

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007

Getting Started With the Code Composer Studio Tutorial

Page 234 sur 548

Address Range	Symbol Name	SLR	Estimated Cycle..	Estimated Start...	Advice Count	Advice List
Loop #1	DolLoop	9-14-DolLoop.c	64	0	2	(2) Options

Notice that the [Estimated Cycles Per Iteration](#) for the loop is currently 64 cycles, and the [Estimated Start Cycles](#) is zero. Also note that the loop has two pieces of Options advice, as indicated by the Advice Count column in the Profile Viewer. For information on the columns in Profile Viewer, consult the online help file: .

The default row height in the profile viewer only allows you to see one advice type in the Advice List column. To increase row height, place the mouse pointer on the solid horizontal line at the bottom of the top row in the Address Range column. Double-click on that line, and the row height will automatically resize to fit all the information.

In the next lesson, you will open the Advice window and analyze the Compiler Consultant advice.



### Options Advice Implementation

CCT - LI

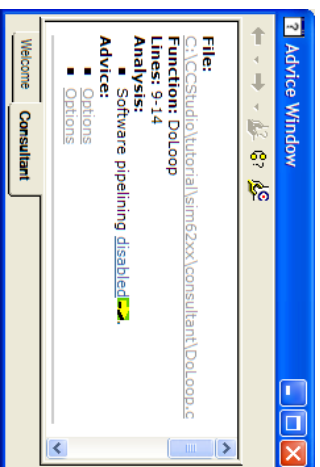
In this lesson, you will analyze the options advice given by the Consultant Tool, and implement it for the consultant.pjt project.

1. Double click on the Advice List cell for the DolLoop row.


The Advice Window appears, and the Consultant tab displays advice for the DolLoop function. The Analysis of the DolLoop functions indicates that the software pipelining optimizations in the compiler are currently disabled. Also note the two pieces of Options advice.

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007



2. Click on the link to the first piece of Options advice.

The first piece of Options advice states the DoLoop is missing the -o compiler option, thus, we are compiling without optimization. The word optimization appears underlined and in color, similar to a link in a typical HTML document. In Code Composer Studio, compiler terminology in advice is often linked to the online help, as indicated by the online help icon .

The suggested solution indicates you should turn on the -o2 or -o3 compiler option. Before doing so, let's take a quick look at the second piece of advice.

3. You may return to the top of the Advice Window by scrolling back to the top with the scroll bar.
4. Click on the link to the second piece of Options advice in the advice list.

The second piece of Options advice states that we are compiling with the -g debug option turned on. As the advice notes, compiling with -g hurts performance. The advice suggests turning off the -g option.

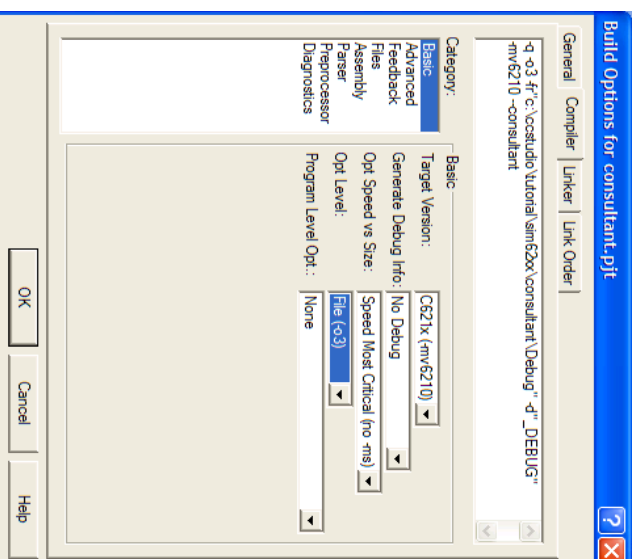
5. Under the Project menu, choose Build Options...
6. In the Build Options dialog, click the Compiler tab.
7. Click on the Basic item in the Category list.
8. From the Generate Debug Info drop down list, choose No Debug.
9. From the Opt Level drop down list, choose File (-o3).
10. Your build options should be similar to:

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007

Getting Started With the Code Composer Studio Tutorial

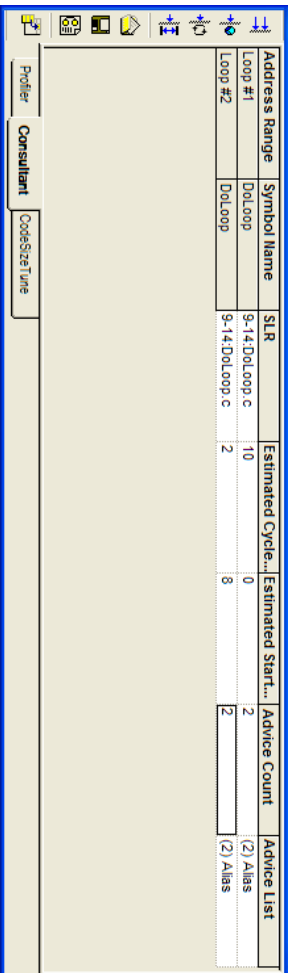
Page 236 sur 548



11. Click OK to close the Build Options dialog.
12. From the Project menu, choose Build.
13. After the build, the advice in the Advice Window is no longer valid, so it clears the advice and informs the user that double clicking on the Advice Count column will refresh the advice.
14. In the Profile Viewer, select the Consultant tab if it is not already selected. Double-click on the Advice Count cell for the Loop with an Estimated Cycles Per Iteration of 10 to update the Advice Window for this loop.

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007



Address Range	Symbol Name	SLR	Estimated Cycle...	Estimated Start...	Advice Count	Advice List
Loop #1	DolLoop	9-14DolLoop.c	10	0	2	(2) Alias
Loop #2	DolLoop	9-14DolLoop.c	2	8	2	(2) Alias

There are now two pieces of Alias advice for the main loop in the profile viewer. Before looking at the specifics on these advice topics, the next lesson will cover loop analysis.



### Loop Analysis

CCT - LI

Look at the Consultant tab of the Advice Window. The Analysis for this loop indicates it has been duplicated. Loop duplication can occur when the compiler is unable to determine if one or more pointers are pointing at the same memory address as some other variable. Such pointers are called **aliases**. One version of the loop presumes the presence of aliases, the other version of the loop does not. The compiler generates code that checks, at runtime, whether certain aliases are present. Then, based on that check, the compiler executes the appropriate version of the loop. The copy of the loop presently displayed in the Advice Window executes when the runtime check for aliases indicates that aliases may be present.

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007

Getting Started With the Code Composer Studio Tutorial

Page 238 sur 548



1. Double-click on the Advice Count cell in Profile Viewer for the other row.

Look at the analysis of that loop. It has a similar note about a duplicated loop, but it presumes certain aliases are not present.

This pseudo-code summarizes the relationship between the three variations of the loop.


```
if (aliases are present)
    Loop #1
else
    Loop #2
```

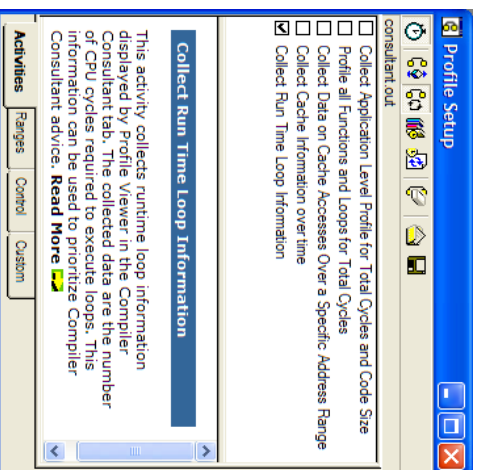
We can profile the code to see which loop versions get selected when the code is executed.

2. From the File menu, choose Load Program to start the program load.
3. Browse to C:\CCStudio\_v3.10\tutorial\sim62xxx\consultant\Debug.
4. Choose consultant.out.
5. Click Open.
6. From the Profile menu, choose Setup.

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007

7. Select the Activities tab near the bottom of the window.
8. The Enable/Disable Profiling icon  should be toggled on, if it is not toggled on, click on it to enable profiling.
9. Place a check next to Collect Run Time Loop Information.



This activity will count the cumulative total of CPU cycles, ignoring system effects, executed in each of the 3 versions of the loop. System effects that it ignores include things like cache misses, memory bank conflicts, off-chip memory access latency, etc. The abbreviated name for this statistic is cycle.CPU:Excl.Total.

- Note:** Your execution platform may not support collection of the event cycle.CPU, and so will not show the activity Collect Run Time Loop Information. In that case, choose the activity Profile All Functions and Loops for Cycles instead. This activity also counts the cumulative total of the CPU cycles, but includes system affects. Further, in addition to profiling all versions of the loop, it profiles each function. In the sections which follow, use the column cycle.CPU:Excl.Total in place of cycle.CPU:Excl.Total. For more information on runtime profiling events access, see the online help file: .
10. Run the program by choosing Debug→Run.
  11. Wait until the program halts. One indicator is the message Halted in the lower left corner. Select the Consultant tab in Profile Viewer if it is not already selected.

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007

## Getting Started With the Code Composer Studio Tutorial

Page 240 sur 548

12. Hover the mouse over the column that starts with cycle.CPU to see that the full name of the column is "cycle.CPU:Excl.Total".

Address Range	Symbol Name	Estimated Cycle...	Estimated Start...	Advice Count	Advice List	cycle.CPU:Excl. ....
0:0x4414-0x4440	D0Loop	10	0	2	(2) Alias	0
0:0x4480-0x44b8	D0Loop	2	8	2	(2) Alias	104

Note the values in this column are 0 for the loop that presumes aliases, and 104 for the other. The runtime check concluded that aliases are not present, and the loop that presumes no aliases is executed.

```
if (aliases are present)
    /* does not execute */
    loop #1
else
    /* execute because runtime check finds no aliases */
    loop #2
```

There are only 2 loops to consider in this simple application. This is unlikely to happen in a real application. Typically, information from many loops is present, and sorting the loops can help identify potential problems. You can sort on any column by double-clicking on the column header.

13. Double click the cycle.CPU:Excl.Total column header multiple times to sort the rows, toggling between ascending and descending order.  
We will now focus on the executed loop. As we implement the alias advice for this loop, the compiler will no longer generate the other version of the loop, and it will no longer appear in the Profile Viewer.
14. Double click on the Advice Count call for the executed loop.

Notice that the Estimated Cycles Per Iteration for the loop has been reduced from 64 cycles to 2 cycles, while the Estimated Start Cycles is now 8 cycles.  
There are now two pieces of Alias advice.



file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007



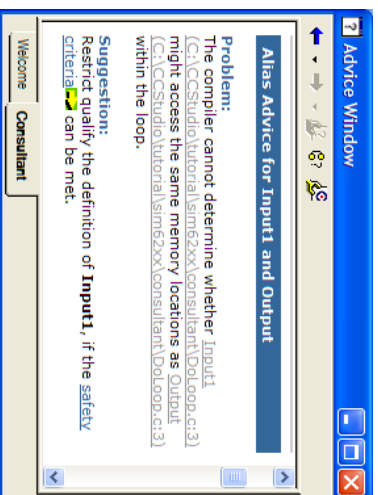
## Alias Advice Implementation

CCT - L1

In this lesson, you will analyze the alias advice given by the Compiler Consultant Tool, and implement it for the consultant.plt project.

1. In the Consultant tab of the Advice Window, click on the link to the first piece of Alias advice.

The problem statement indicates the compiler cannot determine if two pointers may point to the same memory location, and therefore cannot apply more aggressive optimizations to the loop. The Consultant Tool offers several suggestions to solve the problem.



The primary suggestion tells us to use the **restrict** qualifier on the Input1 pointer if the **safety criteria** can be met. Inspect the call to the function DoLoop, and notice that the array passed in as the argument for Input1 is the global array also named Input1. This array does not overlap any of the arrays passed in for the other arguments. This means that the memory referenced by the Input1 variable cannot be referenced by any other variable in the function. Therefore, the Input1 variable meets the safety criteria required for correct use of the restrict keyword. Before we apply the primary suggestion, let's take a look at the alternate suggestions.

The first alternate suggestion indicates that using the restrict qualifier on the Output pointer could solve the problem. Like Input1, we know that Output meets the safety criteria required for use of restrict. Finally, the second and third alternate suggestions indicate that restrict qualifying a local copy of the Output or Input1 pointer would also solve the problem.

2. Go back the top of the advice window and click the link to the second piece of Alias advice.

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007

## Getting Started With the Code Composer Studio Tutorial

Page 242 sur 548

This piece of alias advice is similar to the first, except it mentions pointers Output and Input2.

As there are two pieces of alias advice, one for Output and Input1, and the other for Output and Input2, restrict qualifying Output resolves both pieces of advice as indicated by the example code link in the advice.

3. Click on the link that says Output in the Problem section of the Advice Window to open the DoLoop.c file on the line that declares Output as an argument passed in to the DoLoop function.
4. Modify the DoLoop.c to remove the C comment delimiters `/**/` around the restrict qualifier to the Output pointer parameter.
 

Before:

```
void DoLoop(short *Input1, short *Input2, short /*restrict*/ Output, short *Weights, int LoopCount)
```

After:

```
void DoLoop(short *Input1, short *Input2, short *Weights, int LoopCount)
```

```
#include "DoLoop.h"
...
void DoLoop(short *Input1, short *Input2, short * restrict| Output,
...
{
    int i, Vector1, Vector2;
    short Weight1 = Weights[0];
    short Weight2 = Weights[1];
    for (i = 0; i < LoopCount; i++)
    {
        Vector1 = Input1[i] * Weight1;
        Vector2 = Input2[i] * Weight2;
        Output[i] = (Vector1 + Vector2) >> 15;
    }
}
```

5. Save the file with menu item File→Save, or Control+S.
6. In the Project View, open the include folder and double-click on the DoLoop.h file.
7. Remove the C comment delimiters `/**/` around the restrict qualifier to the Output pointer parameter.

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007



Before:

```
void DoLoop(short *Input1, short *Input2, short */*restrict*/ Output, short *Weights, int LoopCount);
```

After:



```
void DoLoop (short *Input1, short *Input2, short * restrict Output
```

8. Save the file with menu item File→Save, or Control+S.  
Source changes are complete, so it is time to build. However, building the project removes all the current Profile Viewer data, because it will be invalid. Therefore, we should save the data before that occurs.
9. In the Profile Viewer toolbar on the left, click the icon  for Save Current Data Set.
10. Browse to C:\CCStudio\_v3.10\tutorial\sim62x\consultant\Debug.
11. Enter runt for the file name.
12. Select Dataset (\*.xml) for the Save As type, and click Save.  
To see this data again later, you can use the Load Data Set  button in the Profile Viewer.
13. From the Project menu, choose Build.

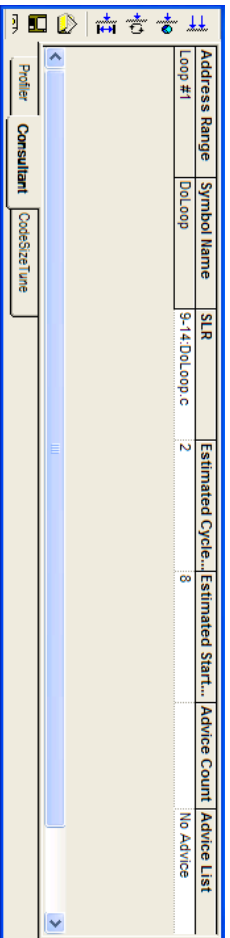
file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007

Getting Started With the Code Composer Studio Tutorial

Page 244 sur 548

14. In the Profile Viewer, select the Consultant tab if it is not already selected. Double-click on the Advice Count cell for DoLoop to update the Advice Window for this loop.



Address Range	Symbol Name	SLR	Estimated Cycle...	Estimated Start...	Advice Count	Advice List
Loop #1	DoLoop	9-14.DoLoop.c	2	8	No Advice	

The estimated cycles per iteration remains at 2, and the estimated start cycles remain at 8. However, the duplicated loop which presumes aliases and the runtime check for aliases are both removed. This is a small savings in cycles, and a sizeable savings in code size.

The Advice Count is now 0, and we have reduced the total cycle time from an initial value of  $64 * 40 = 2560$  cycles to  $8 + (2 * 40) = 88$  cycles, or roughly a 29X improvement.

Throughout this tutorial we have been applying compiler optimization changes that not only affected performance, but also affected the code size. Code size versus performance is a trade-off that only the designer can make. To help you evaluate code size versus performance for the DoLoop function, you can use the [CodeSizeTune tool](#).

For more information, see the application note *Introduction to Compiler Consultant*.



**CacheTune Introduction**

**CACHE - Intro**

The complexities of DSP (Digital Signal Processor) applications and their system memory, utilization require the user to optimize the application to make efficient use of the memory subsystem to meet performance goals. Efficient use of on-chip caches improves the CPU throughput by reducing CPU stall cycles due to memory activity.

The CacheTune tool provides graphical visualization of memory reference patterns for program execution over a set amount of time. All the 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 greatly improve the cache efficiency of the overall application.

#### Learning Objectives:

- Prepare an application to collect cache data

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007

- Assess the cache overhead for the application
- Visualize the memory accesses in cache
- View the symbolic information
- Use the zooming features to view cache information
- Use Interference grid, Interference shadow, and Find Conflicting Symbols tools to identify object conflicts
- Eliminate the conflict misses in the program cache
- Measure the improvement

**Example used in this lesson: Matrix Transpose Operation (mat\_oprn)**

Target Configuration: The Matrix Transpose Operation demonstration application is located in the `C:\CCStudio_v3.10\tutorial\sim64xx\mat_oprn` directory. To run this tutorial, you must setup the Code Composer Studio™ IDE to select the C6416 device cycle accurate simulator, little endian. For C62xx simulators, you can find the example in the `sim62xx\mat_oprn` directory. For more information about CCSStudio Setup, run Setup using the Setup CCSStudio icon, and use the online help. Before using this tutorial module, you should have done the following:

- Installed the Code Composer Studio™ IDE
- Set up the Code Composer Studio™ IDE to run on the C6416 device cycle accurate simulator, little endian.
- Familiarized yourself with C6000 two-level memory architecture and cache related terminologies (see online help for more information)

**Application Objective:**

This tutorial introduces the basic features of CacheTune and takes you through a step-by-step process for analyzing and optimizing an application's cache performance using CacheTune. The demonstration example performs the following operation on matrices A and B and stores the result in C.

```
C = A <mat_oprn> * transpose (B)
```

It highlights the capacity misses in the data cache and conflict misses in the program cache. It also recommends cache optimization techniques to eliminate these misses.

This lesson consists of the following steps. To go directly to a step, click on the link below:

[Getting Started](#)[Data Cache](#)[Program Cache](#)

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007

Getting Started With the Code Composer Studio Tutorial

Page 246 sur 548



The forward arrow will take you to the next page in this lesson.

**Opening and Reviewing the Project****CACHE - L1**

You begin this lesson by opening a project and examining the source code files used in the project. This lesson uses the `mat_oprn` demonstration application.

1. Create a folder called `mat_oprn` in the `C:\CCStudio_v3.10\MyProjects` folder.
2. Copy all the files and folders from the `C:\CCStudio_v3.10\tutorial\sim64xx\mat_oprn\mat_oprn_1` folder to this new folder.
3. If you have not already done so, from the Windows Start menu, choose Programs→Texas Instruments→Code Composer Studio 3.1→Code Composer Studio 3.1. (Or, double-click the Code Composer Studio icon on your desktop.)
4. If another project was already open in Code Composer Studio, right-click on that project's .pjf file in the Project View and choose Close from the context menu. From the Project menu, choose Open.
5. Browse to `C:\CCStudio_v3.10\MyProjects\mat_oprn`.
6. Select `mat_oprn_1.pjt` and click Open to load the project.
7. Expand the Project View list by clicking on the plus (+) signs next to Project, `mat_oprn_1.pjt`, and Source.
8. Double-click on the `mat_xpose_oprn.c` program to open it. Examine the source code for this program. Function `mat_xpose_oprn` can be found in this file. It calls function `mat_oprn` to perform the operations on matrices A and B. Function `mat_oprn` operates on the `i, j`-th entry of matrix A and `j, i`-th entry of matrix B. The result is stored into the `i, j`-th entry of matrix C. Notice that matrices A and C are accessed in row major order whereas matrix B is accessed in a column major order. CODE\_SECTION pragma is used to place the function in its own section.  
[Source Code](#)
9. Double-click on the `mat_oprn.c` program to open it. Examine the source code for this program. Function `mat_oprn` can be found in this file. It calls calculate function and performs several operations on the elements in the matrices and returns the results. CODE\_SECTION pragma is used to place the function in its own section.  
[Source Code](#)
10. Double-click on the `mat_xpose_oprn_1.cmd` program to open it. Examine the source code for this program. Notice that, by a relative placement in the linker command file, the two functions: `mat_oprn` and `mat_xpose_oprn` are placed into the memory map such that they map to the same cache line in the level-one program cache.

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007

[Source Code](#)

The example is demonstrated within TMS320C64x™ (C64™) environment. The C64x™ DSP family has a dedicated level-one program cache (L1P) and data cache (L1D) of 16 Kbytes each. L1P is a direct-mapped cache with a 32-byte line size. L1D is a 2-way set associative cache. Each cache way is 8K bytes and the cache line is 64 bytes. This information is necessary to understand the cache behavior with this particular example.


In the next lesson, you will assess the cache overhead.



### Assessing the Cache Overhead

#### CACHE - L1

In this step, you will profile your application to assess the cache related overhead, using the profiling features. The cache overhead can be assessed by measuring the cache stall cycles caused predominately by cache misses and L1D write buffer full occurrences. For more detailed discussion on cache overhead, please refer to the related CacheTune online help topic.

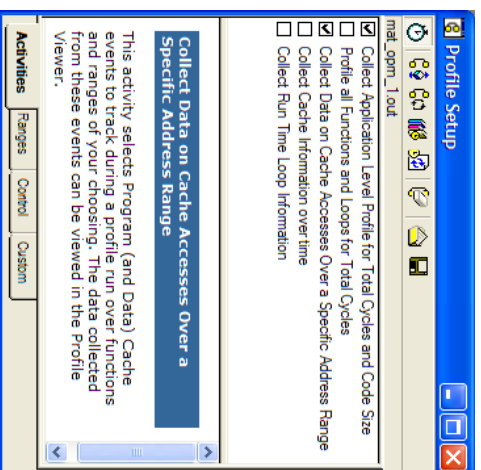
1. From the Project menu, click on Rebuild All. An output file (.out) must be generated and loaded before beginning the profiling process.
2. From the File menu, click on Load Program, and choose mat\_oprn\_1.out from the Release subfolder. Click Open.
3. From the Profile menu, launch Setup. The Profile Setup opens on the right side of the screen and allows the setting of several options to customize application profiling. The Profile Setup window displays the activities tab by default, to allow users to select the profiling collection activities.
4. Check the icons at the top of the Profile Setup tool to determine if profiling is enabled. The Enable/Disable Profiling icon  must be toggled on to enable profiling. Profiling may not be enabled if the program has started running or if it was explicitly disabled. Data will not be collected if profiling is not enabled.
5. Click on the box beside Collect Application Level Profile for Total Cycles and Code Size. This activity measures the total cycles consumed by the entire application and calculates the total code size of the application. This information can be viewed later from Goals Window.
6. Click on the box beside Collect Data on Cache Accesses Over a Specific Address Range. This activity tracks Program and Data Cache events, such as cache stall cycles, and the number of cache misses, during a profile run over functions and ranges of your choosing.

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007

Getting Started With the Code Composer Studio Tutorial

Page 248 sur 548



7. Select the Profile→Tuning→Goals menu item to open Goals Window. Please see the [Goals Window](#) lesson for more information.
8. Select Profile→Viewer menu item to open Profile Data Viewer. Please see the [Profile Viewer](#) lesson for more information.
9. Select the Debug→Run menu item or press F5 to run the application. The program self-terminates, so it does not need an exit point to stop profile data collection. Please see the online help topics on [profile setup](#) for more information about exit points. Once the program terminates, you may view the cache accesses in the Profile Viewer.
10. Once the program has executed, initial data values will appear in the Current column of the Goals window. These values represent the Code Size and Cycle Total for this tutorial, we will mainly focus on the cycle total and use the Goals Window to track tuning progress for improvements in the cycle total. The total cycle count is 1,348,344. Your numbers may vary.

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007

Goal	Current	Previous	Delta
Code Size	1696	-	-
Cycle Total	1348344	-	-

- After the program finishes execution, a new Summary tab will be added in the Profile Viewer. The Summary tab displays the total counts and percentages of most available simulation events. Please refer to the Dashboard [Profile Viewer](#) online help for a detailed description of the displayed events for different targets.
- In the Profile Viewer window, click on the Summary tab and look at the number for the following events. These are sample numbers, your numbers may vary.
  - Total Cycles: (1,347,9265 cycles)
  - Core Cycles (excl. stalls): (738,749 cycles, 59.12%)
  - L1P Stall Cycles (510,199 cycles, 40.83%)
  - L1D Stall Cycles (99,397 cycles, 7.95%)

**Tip:** You may roll your mouse over the event cell to display the complete name.

In the next lesson, we will collect data on cache accesses over time with Profile Setup.



### Collecting Cache Access Data

CACHE - L1

To collect data on cache accesses, use Profile Setup to select the appropriate activity by using the following steps:

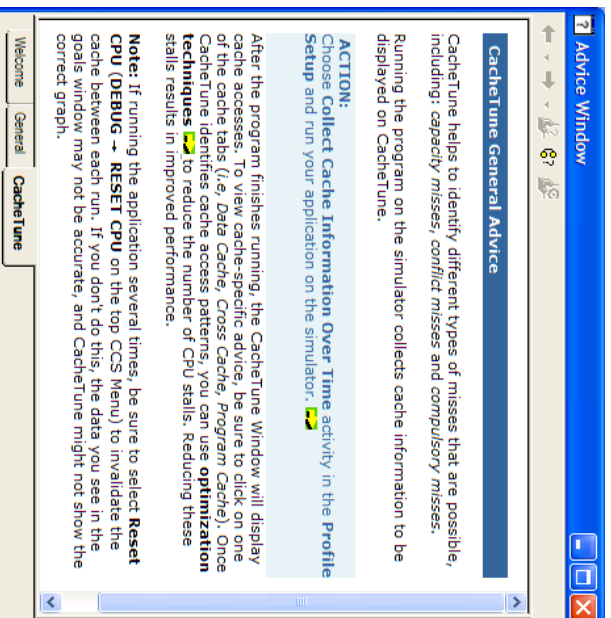
- From the Profile menu, launch Tuning->CacheTune. This selection launches the CacheTune tool in the main editor window. It also launches the advice window and activates the CacheTune tab. The page describes the tool, and provides suggestions for getting the most out of the tool. At this point, the program has not collected any cache data over time, so most of buttons on the toolbar are deactivated.

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007

Getting Started With the Code Composer Studio Tutorial

Page 250 sur 548

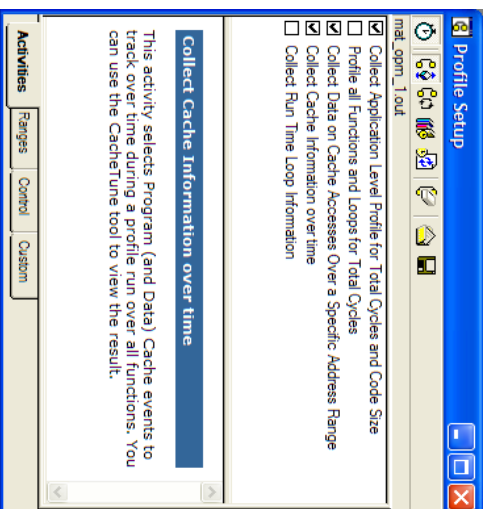


Note that you are directed to collect cache data by selecting the appropriate cache-related activity in Profile Setup.

- From the Profile Setup window, select the activity Collect Cache Information over time. This activity selects cache events to track over time during a profile run over all memory addresses.

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007



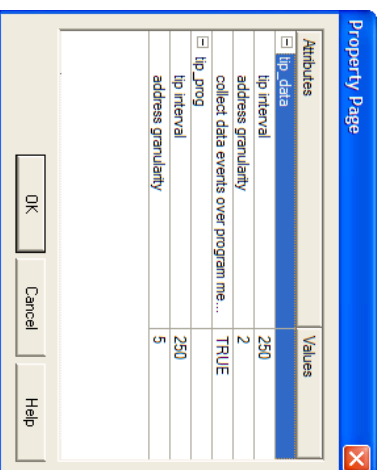
- Right-click on the selected activity, and select the Property Page from the context menu. This opens the Property Page, where you can modify the attributes of trace data, such as time interval profile (tip) interval and address granularity. The tip interval is the size in cycles of the time period over which the simulator summarizes the memory references before writing out a trace record. The address granularity for 'tip\_data' is 2 bytes, meaning that any access to a byte is visualized as if it were to a half word.

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007

Getting Started With the Code Composer Studio Tutorial

Page 252 sur 548



- The default tip interval is 250 cycles for the program and data caches. Left-click in the first tip interval cell that is for 'tip\_data' to activate that cell. Once activated, change the cell value to 10, so you will have a larger resolution and, therefore, view more details on cache access for the data cache. Please refer to the related topic of Cachetune online help for more details.
- Click OK to save changes and close the Property Page.
- Before we run the program to collect data on cache accesses, we need to perform a CPU reset to clear the cache, as suggested by the General Advice Window for our second run of the program. From the Debug menu, choose Reset CPU.
- Select File→Reload Program to reload the program
- Now we are ready to collect data on cache accesses. From the Debug menu, choose Run, or press the F5 key. The program will run to completion.

Once the program terminates, you may view the cache accesses from Cachetune.

In the next lessons, you will analyze the data cache with the Cachetune tool and eliminate the cache misses to improve the overall performance.



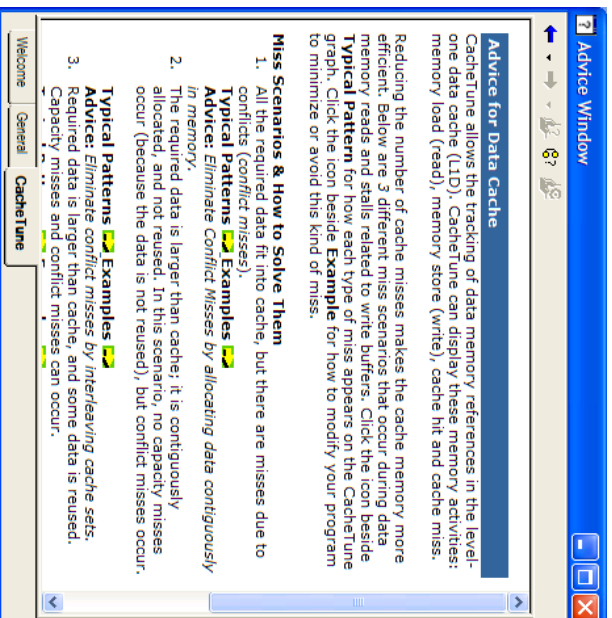
**Preparing to View Data Cache Accesses**

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007

In this exercise, you will use CacheTune to visualize and analyze the memory access patterns in the data cache and view the available symbolic information. In addition, you will learn to navigate within the tool and view a particular area using the zooming features. We will also discuss the optimization techniques used to eliminate the cache misses in the data cache.

After the program terminates, navigate to the CacheTune output window in your workspace to view memory accesses. By default, CacheTune displays the data cache tab. Click on the data cache tab title to display data cache advice in the advice window. If you have multiple windows open in the main CCS workspace, they may cover the CacheTune output graph window after profiling is completed. In this case, close some windows to view the output graph window.



**Note:** When tuning, always start with data cache, because optimizing for data cache performance may lead you to modify your program, thus changing the memory access pattern of the program cache.

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007

Getting Started With the Code Composer Studio Tutorial

Page 254 sur 548



**Viewing the Data Cache Accesses**

CACHE - L2

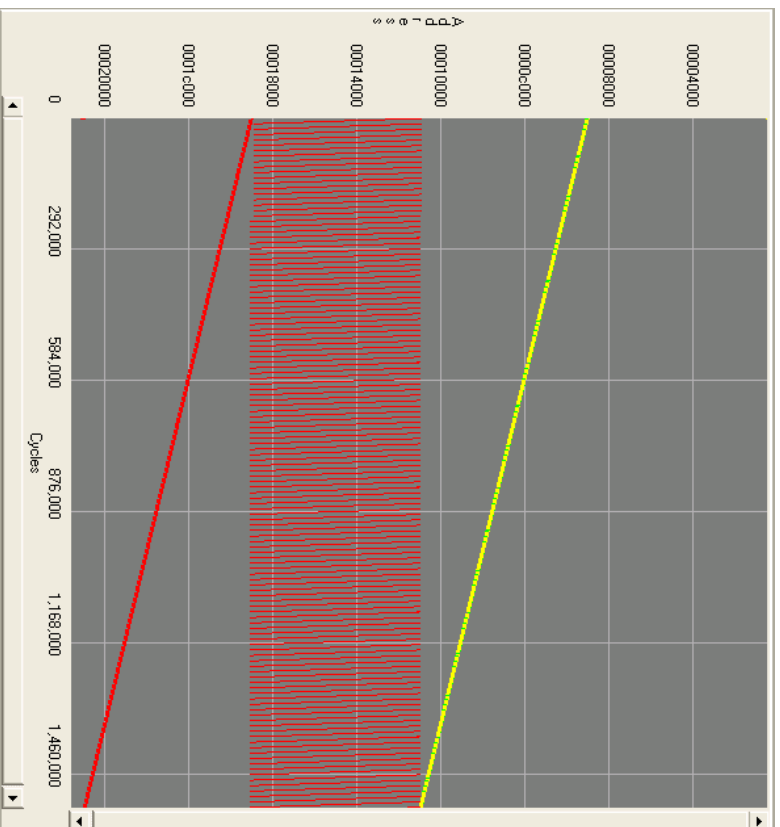
CacheTune graphically shows memory access patterns over time, color-coding the accesses to distinguish cache hits from cache misses. Cache hits are shown as green pixels while cache misses are shown in red. The accessed addresses are plotted along the Y-axis, while access times are plotted along the X-axis.

**Viewing the Data Cache Accesses**

1. Adjust the CacheTune window to a suitable size. This view only shows a small part of the cache trace data (indicated by the horizontal and vertical scroll bar).
2. Click on the Full Zoom button to view the entire trace. This provides an overview of memory access patterns and points out the hot spots where most cache misses occur. The graph should resemble the image below. There is a red bar in the middle of two inclined lines. The line on the top is partially yellow and green; the bottom is red. This indicates that both cache misses and hits occur when accessing the memory range associated with the upper line; the accesses to the middle and lower part of the memory addresses incur intensive cache misses.

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007



- Move the cursor across the display. The information display area (left side of the CacheTune window) updates with the current address range, section, symbol (if any), and cycle range corresponding to the pixel currently under the cursor. This provides the information of where (address) and when (cycle) cache events occur on which data (section and symbol) memory access. As observed from the display, cache events occur on memory accesses to matrices A, B and C through almost the entire program. The memory ranges are 0x90000 to 0x11000, 0x11000 to 0x19000, and 0x19000 to 0x21000 respectively. The address range matches the size of each matrix, which is  $0x8000$  bytes. ( $128 \times 128 \times 2 = 32768 = 0x8000$  bytes)

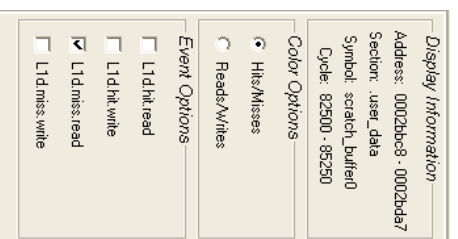
file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007

Getting Started With the Code Composer Studio Tutorial

Page 256 sur 548

- From the Color Options area on the left of the display, switch between the two different color schemes: Hits/Misses and Reads/Writes. One color scheme codes the memory references according to whether they were cache hits or cache misses, the other color scheme codes the references according to whether they were reads or writes. The area above the graph provides a legend for the colors used in the graph for various events. For example, with Reads/Writes color scheme, the cache reads color code in green pixels and cache writes in red. When a pixel is yellow, it indicates both a cache read and write occur. Notice that all accesses to A and B are cache reads and accesses to C are cache writes.
- Reset the color scheme to Hits/Misses.
- With the Hits/Misses color scheme on screen, try toggling some of the cache events on and off from the Event Options area. For instance, un-check other cache events and select L1dmiss:read event to see only the read misses events in the data cache.



- Un-check the other cache events and select L1d hit:read to see only the cache read hits events. Notice that there are no cache hits on accesses to matrices B and C, that is, all accesses to matrices B and C miss the cache.

With the help of features available from the tool, we quickly identify some basic access patterns: The upper part of the trace corresponds to the reading (load) of the input matrices A and B. Then the results are written (store) into matrix C. There are both cache misses and hits when accessing matrix A, whereas all the accesses to matrices B and C cause cache misses.

- Re-check all the Event Options boxes.

Since the hot-spot associated with most cache misses is accessing matrix B, the next step will take a closer look at it to analyze the code behavior and identify the causes of misses.

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007





## Zoom Features

CACHE - L2

### Using the Zoom Features on the Graphical Display Area

CacheTune provides several zoom features to facilitate navigation, and allow the user to identify the cache miss problem areas.



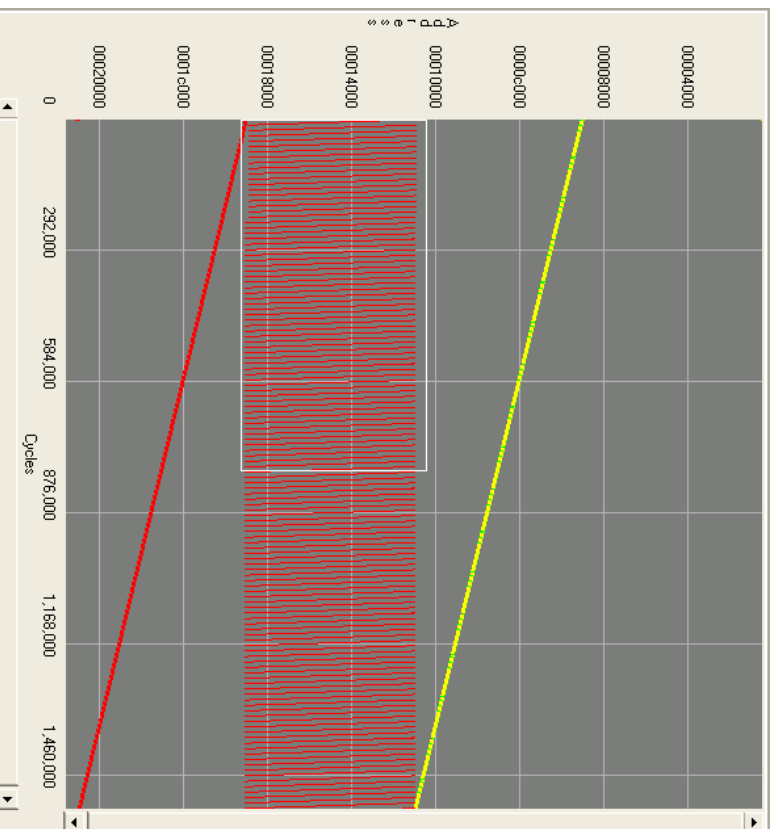
1. Select the **Zoom Area** button. The cursor will change from pointer mode to zoom mode. CacheTune has four different cursor modes: pointer, zoom, grid, and shadow. For more information, please see the online help topic on CacheTune.
2. Left-click to drag a rectangle over a partial area with respect to the accesses to matrix B, shown outlined in white below.

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007

### Getting Started With the Code Composer Studio Tutorial

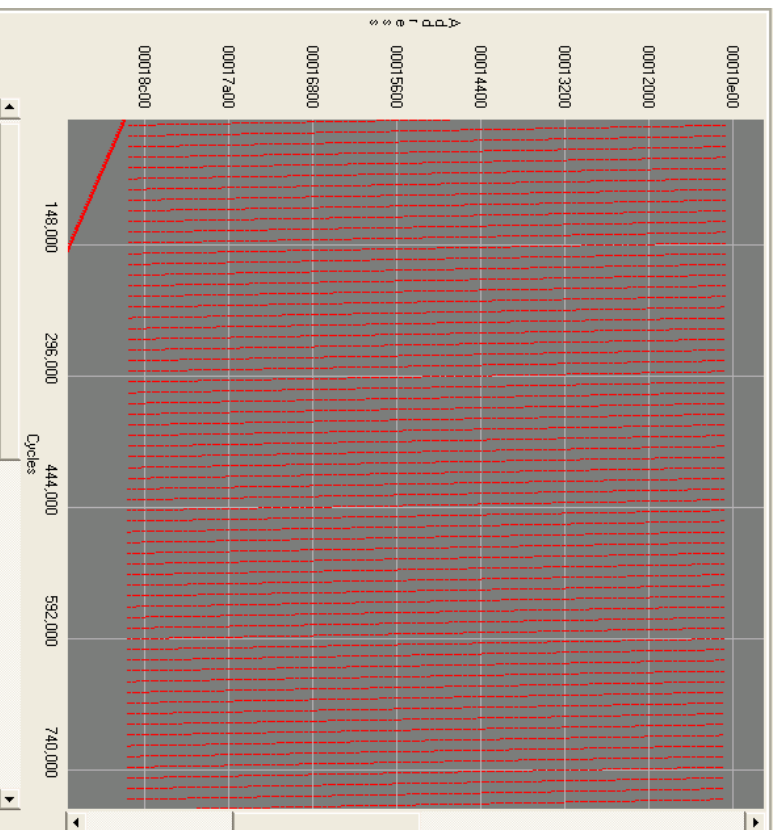
Page 258 sur 548



3. The resulting display should resemble the following image, showing a much more detailed picture of the memory access pattern.

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007



Notice the red bar is composed of multiple red vertical lines because matrix B is accessed by column, that is, the  $n$ -th element of every row is accessed consecutively. The accesses to each column correspond to each vertical red line in the memory range of matrix B on the display. Every two consecutive accesses within a single column are with a stride of 0x100 bytes in memory range. Because the stride is larger than the cache line size of L1D (64 bytes), the accesses cannot take advantage of spatial locality, thus causing cache misses. Furthermore, as the size of matrix B is twice as large as the L1 data cache size, the rows of the matrix B that are already in the cache will be replaced by the rows that are brought into the cache later. This causes cache misses when previous allocated rows are accessed again. By going through the entire dimension of the large matrix, the cache cannot hold the


file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007

Getting Started With the Code Composer Studio Tutorial

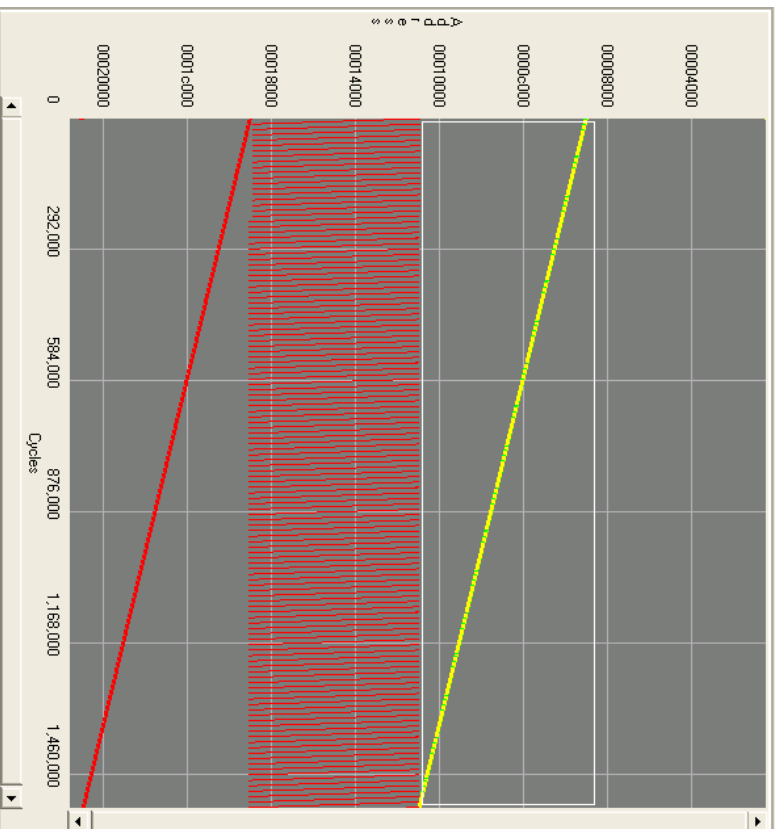
Page 260 sur 548


data that are referenced within particular time period and hence cause capacity misses.

4. Click on the Undo-Zoom  button to reset the display.
5. Left-click to drag a rectangle over a partial area with respect to the accesses to matrix A as shown outlined in white below.

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007



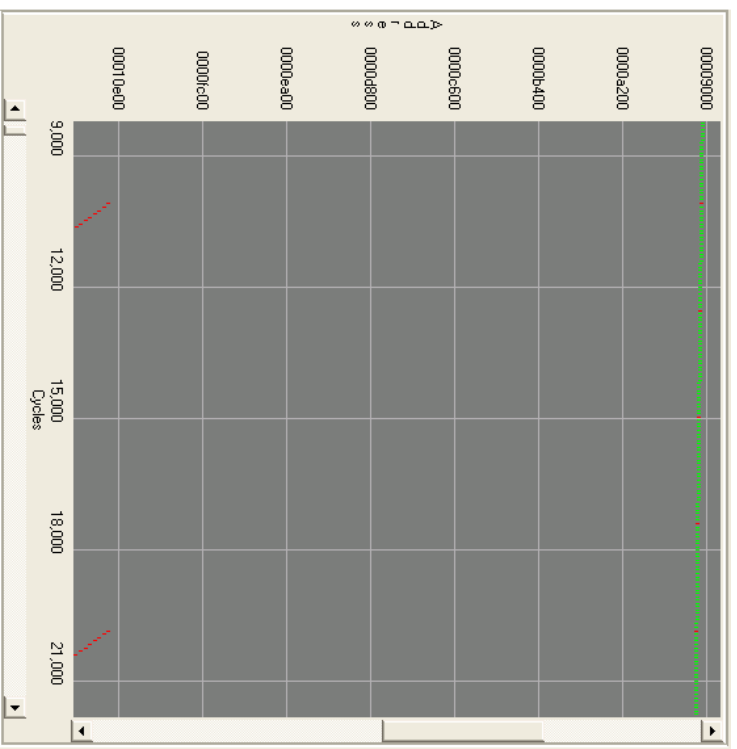
- Click on the Zoom-In Cycles  button to zoom to the cycles (X-axis) only. Repeat this until the graph resembles the image below.

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007

Getting Started With the Code Composer Studio Tutorial

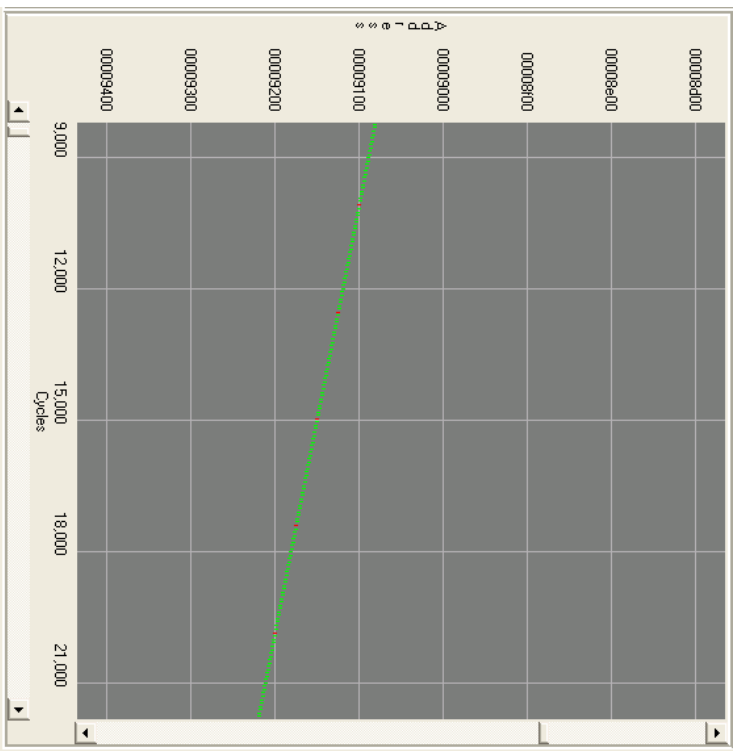
Page 262 sur 548



- Click on the Zoom-In Address  button to zoom to the addresses (Y-axis) only. Repeat this process until the graph resembles the image below.

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007



The program makes a large series of sequential accesses to matrix A. The access pattern of matrix A is demonstrated in the above image. When a cache miss occurs, 32 matrix elements, which are 64 bytes of data (size of cache line in L1D), will be brought into the cache. Since elements in each row of matrix are accessed consecutively, there are 31 consecutive cache hits after one cache miss as can be observed. The cache misses here are mostly compulsory misses.

The memory access pattern of matrix C is almost identical with matrix A, except matrix C is accessed by store rather than load operations and all misses are store misses. As we know from the profiling results from lesson 1, there are no write buffer full related stalls for this application. This means the write misses when accessing matrix C do not stall the CPU.

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007

Getting Started With the Code Composer Studio Tutorial

Page 264 sur 548

Now we know that the capacity misses occurred when accessing matrix B during load operations. This matches the third misses' scenario discussed from CacheTune Data Cache Advice Window. As suggested, these misses can be eliminated by dividing the data into several sets and processing one at a time. However, we have no knowledge of which part of the code references the matrix. For our example, which is a small and simple application, the particular matrix is only referenced within one function. But for some applications, particular data could be accessed by multiple functions. The information contained within the Cross Cache will help us at this point.



### Cross Cache

#### CACHE - L2

The cross cache view in CacheTune encodes a cross-reference of program memory accesses with data cache accesses. The addresses are the same as for the program cache display, but the events displayed are with respect to the data memory reads and writes (loads and stores) contained within those instructions and whether those loads and stores hit or missed in the data cache (L1D).

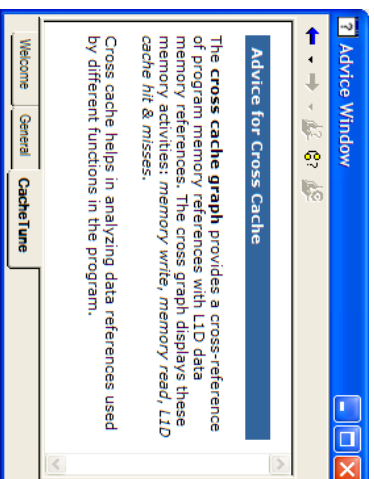
Cross Cache can help to analyze the data memory references by the program. For example, we can tell which functions access the data memory, and whether the accesses miss the data cache or not.

1. Switch to the cross cache view by clicking on the Cross Cache tab located below the CacheTune toolbar. Each tab maintains its own toolbar state, mode, selected symbol, etc. The advice will change to cross cache advice.



file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007



2. Press the Full Zoom  button to view the entire trace.

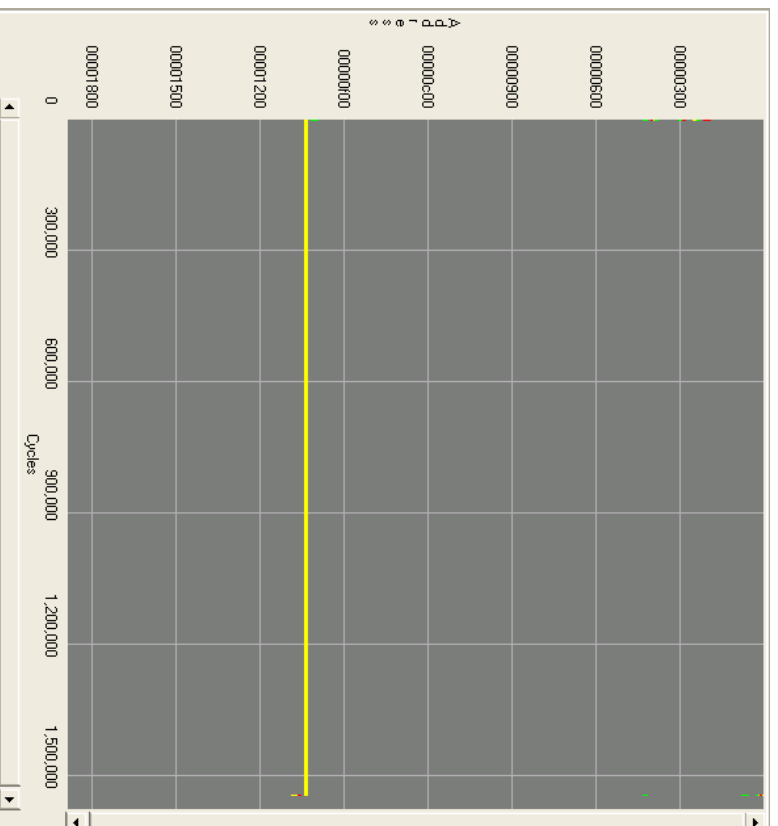
Notice there is one horizontal bar across the display, which represents the code executed repeatedly.

file://C:\Documents and Settings\Pierre-Amand\Local Settings\Temp\~hhBA53.htm

26/09/2007

Getting Started With the Code Composer Studio Tutorial

Page 266 sur 548



3. Place the cursor on top of the bars. The function containing the code then can be easily identified from the Information Display Area. The memory accesses are only associated within function mat\_xpose\_oprn.

In the next lesson, we will optimize the C code.

file://C:\Documents and Settings\Pierre-Amand\Local Settings\Temp\~hhBA53.htm

26/09/2007



## Optimizing C Code

CACHE - L2

Based on the analysis above, most cache misses on L1D are capacity misses due to suboptimal data usage, that is, the algorithm does not reuse the data when they are still in cache. This type of miss can be reduced by restructuring the data in order to work on smaller blocks of data at a time.

1. From Project View window, double-click on mat\_xpose\_oprn.c program to open it.
2. Modify the mat\_xpose\_oprn function as the source code shown below in red. Please include the appropriate tab spaces. The optimized version of source file can be also found in mat\_oprn\_2 directory.

### Before

```
After:
Hexagram CODE_SECTION(mat_xpose_oprn, ".mat_xpose_oprn")
#define MAT_PART_SIZE MAT_ORDER/4
void mat_xpose_oprn(void) {
    short i, j, ii, jj;
    /*
     * Matrices are accessed in parts and the
     * parts are reused
     */
    for (ii = 0; ii < MAT_ORDER; ii += MAT_PART_SIZE){
        for (jj = 0; jj < MAT_ORDER; jj += MAT_PART_SIZE) {
            for (i = ii; i < MAT_PART_SIZE + ii; i++) {
                for (j = jj; j < MAT_PART_SIZE + jj; j++){
                    C[i][j] = mat_oprn(A[i][j],B[j][i]);
                }
            }
        }
    }
}
main()
{
    mat_xpose_oprn();
}
```

The function now works on smaller pieces of the matrix at a time. This reduces the size of the working set and improves temporal locality.

3. Click on the Incremental Build  button to rebuild the only the modified program.

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007

## Getting Started With the Code Composer Studio Tutorial

Page 268 sur 548


4. Choose Debug →Reset CPU to clear the cache.
5. Choose File→Reload Program to reload the program.



## Measuring Cache Usage Improvement

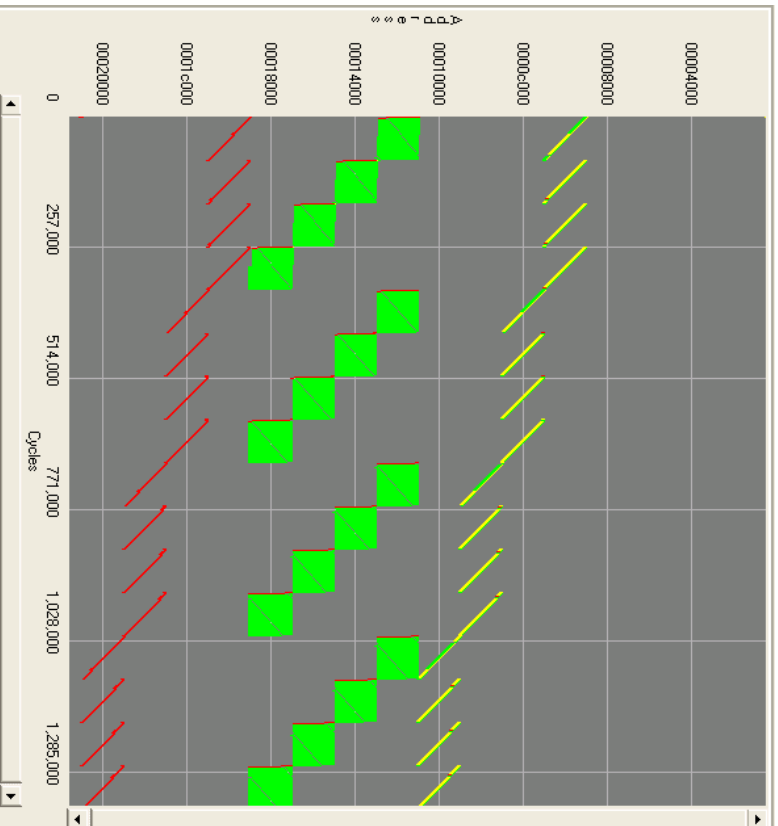
CACHE - L2

We can now visualize the new data cache access pattern with the CacheTune tool and use Goals Window and Profile Viewer to check the improvement.

1. Press F5 key to run the program. Once program finishes execution, the CacheTune output window, Goals Window, and Profile Viewer will update to display the new data.
2. Click on the Data tab of the CacheTune output window.
3. Click on the Full Zoom  button to view the entire trace. Most read misses now are compulsory misses. Also notice that only a smaller block of matrix is accessed within each particular time frame.

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007



4. Check the data in Goals Window. The data from the last execution of the program moves to the Previous column and the difference between values is displayed in the Delta column. Notice that the overall performance of the program improves 143,521 cycles after optimizing the data cache. Your numbers may vary.
5. In Profile Viewer, check the data for the L1D Stall Cycles row in the Summary tab. Now the count is 6,116 cycles, and the percentage is 0.55%, both of which are dramatically improved. Your numbers may vary. These stalls are caused by the compulsory misses when the array elements are first accessed.

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007

Getting Started With the Code Composer Studio Tutorial

Page 270 sur 548

Since there are still a high number of cache misses in the program cache, we need to investigate the program memory access pattern and improve program cache utilization in the next lesson.

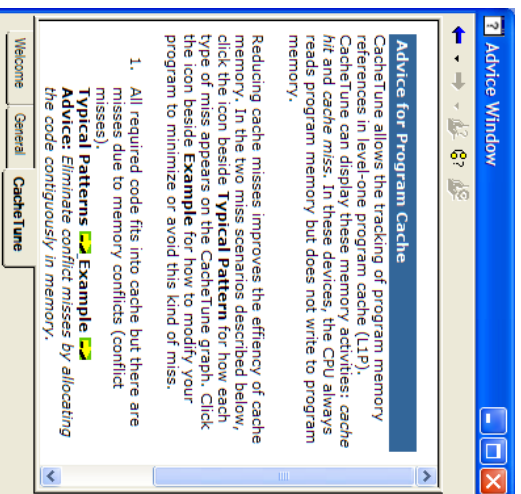


### Program Cache Overview

CACHE - L3

The CacheTune tool is also valuable for improving program memory performance in cache-based devices. The CacheTune display easily highlights conflict misses in the program cache when two or more functions map to the same cache location. In this exercise, you will identify these kinds of conflict misses and learn to use the Interference Grid, Interference Shadow, and Find Conflicting Symbols Tool to locate the conflicting functions. We will also discuss the optimization techniques used to eliminate these misses.

1. Switch to the program cache view by clicking on the Program Cache tab located below the CacheTune toolbar. This also displays the program cache advice in the advice window.

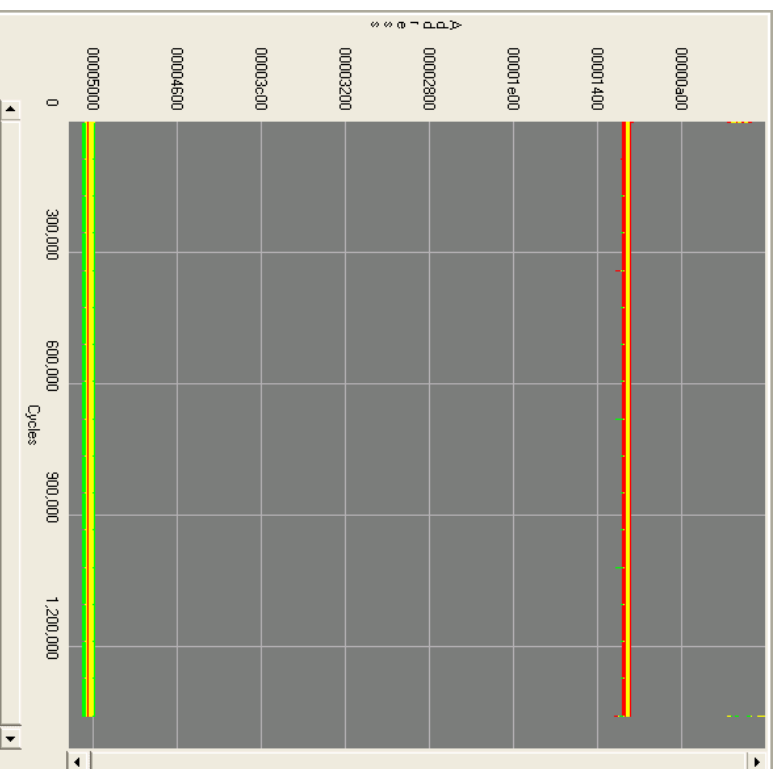


file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007

2. Press the Full zoom  button to view the entire trace.

Note again how the function names change in the Symbol display as you move the cursor up and down over the graphical display area. Notice the horizontal bars across the display. Each bar represents a piece of code that is executed repeatedly at short time intervals. Moving the cursor over the bars identifies the functions that contain the code.



file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

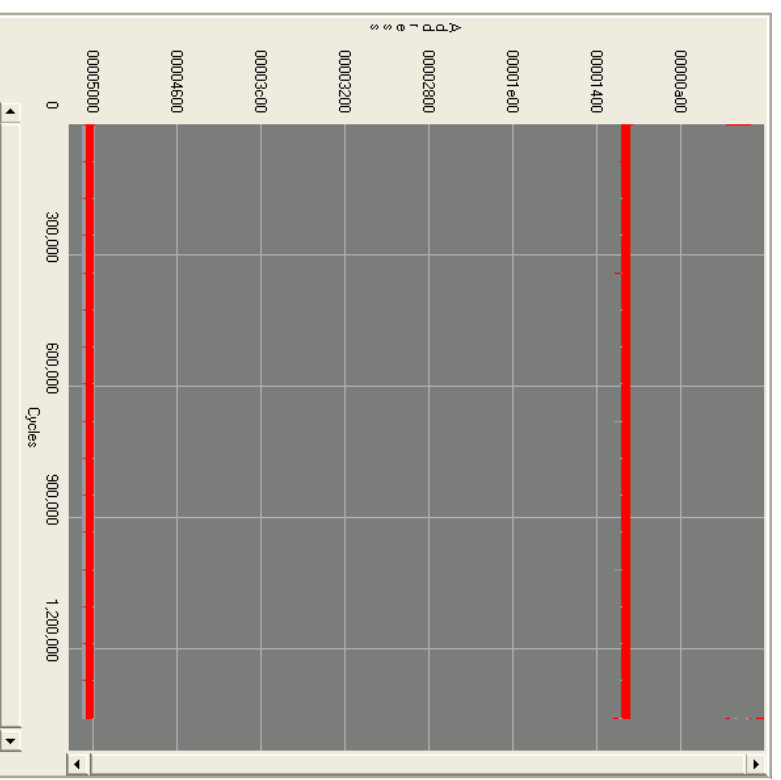
26/09/2007

Getting Started With the Code Composer Studio Tutorial

Page 272 sur 548

3. Roll your mouse over the LIP Misses legend above the graph to view cache misses events only, other events are grayed out.

Notice that there are two red bars along their horizontal extent, signifying that the same pieces of code miss repeatedly in the cache within short periods of time. Cache conflicts cause this type of repetitive misses.



file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007




CacheTune has three tools to help identify which objects (functions in this case) create such conflicts in the cache. These tools are called the Interference Grid, the Interference Shadow, and the Find Conflicting Symbols tools.



### Using Interference Grid

CACHE - L3

The Interference Grid draws a line at every address in the display that interferes with the selected address in the cache.

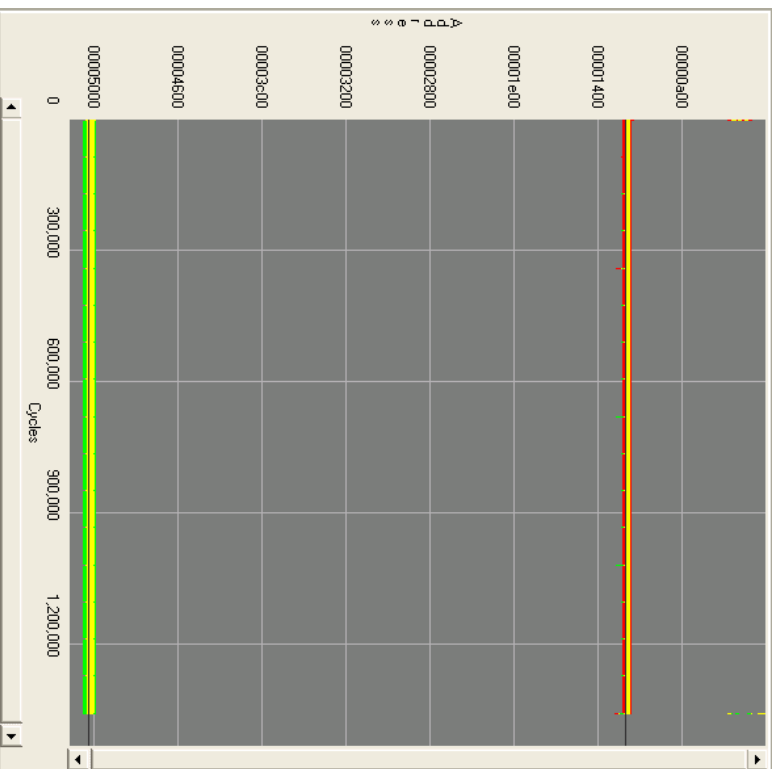
1. Press the **Interference Grid**  button.
2. Select the origin of the grid at an address within the memory range of function mat\_xpose\_oprn (represented by the top horizontal bar). A dark grid will be drawn at any conflicted memory address, which is every other 16K byte address (size of L1P cache) for a direct-mapped L1P on C6416. Notice a dark line is drawn on top of the lower horizontal bar.

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007

Getting Started With the Code Composer Studio Tutorial

Page 274 sur 548



3. Right-click on the graph to clear the interference lines.

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007



## Using Interference Shadow

CACHE - L3

The Interference Shadow draws a shadow at every address range in the display that interferes in the cache with the selected symbol.

**Note:** This feature only works for global symbols that have a size associated with them.

The shadow mode has three different cursors:

- The cursor with two solid bars indicates that it is over a symbol. You can left click to select the shadow.
  - The cursor with two empty bars indicates that no symbols are available under the cursor.
  - The cursor with large filled in box that indicates the cursor is over a symbol conflicting with all of the cache block (larger than cache size). The user can still click this symbol to select it (red name in address area) but the shadow will not be drawn, as it would fill the entire graph.
1. Press the **Interference Shadow** button.
  2. Place the cursor at an address within the memory range of function mat\_xpose\_oprn (represented by the top horizontal bar) and left click. The resulting display will show dark bands

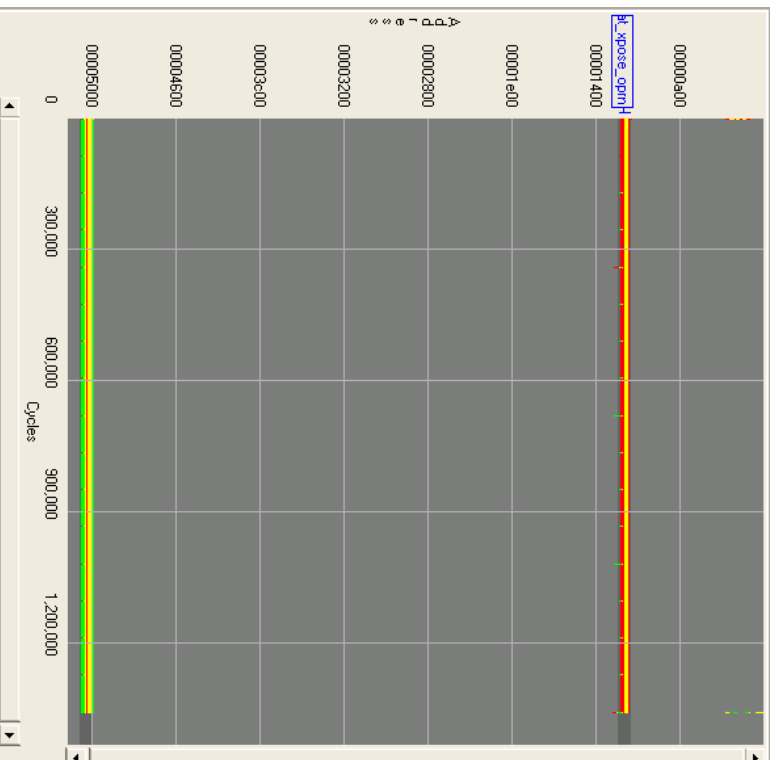
across the graphical display area to indicate the address ranges that interfere with this function in the cache.

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007

## Getting Started With the Code Composer Studio Tutorial

Page 276 sur 548



3. Right-click on the graph to clear the interference lines.


file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007



### Find Conflicting Symbols

CACHE - L3

When you select the Interference Shadow, it will display the symbol name, "mat\_xpose\_oprn" in the Address Section; this also enables the Find Conflicting Symbols  button. Find Conflicting Symbols displays a list of the symbols and graphs that conflict in memory with the selected symbol.

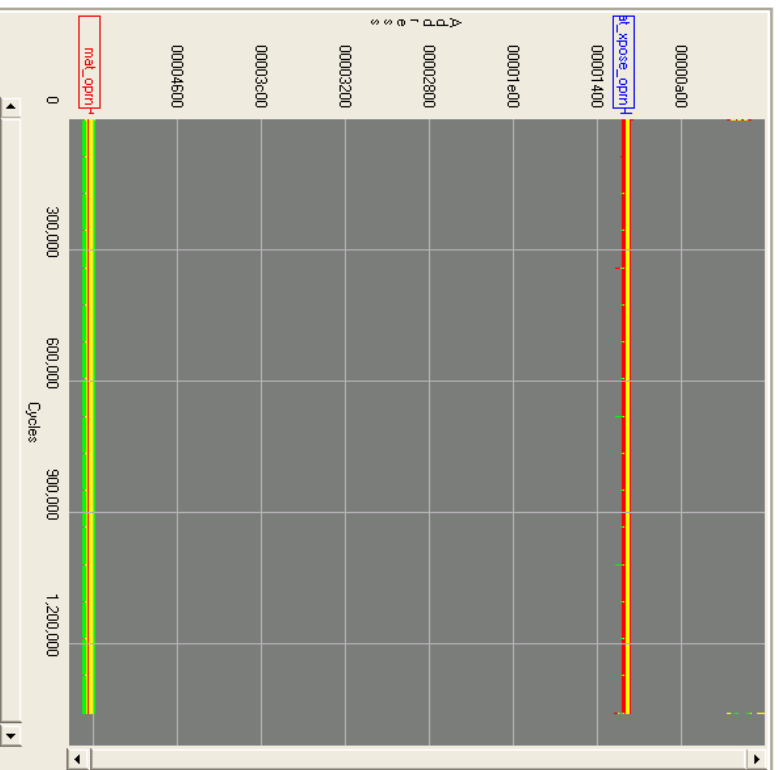
1. Click on the Find Conflicting Symbols button, . This displays all cache conflicting symbols with a red box around them in the Address Section.

file:///C:/Documents and Settings/Pierre-Armand/Local Settings/Temp/~hhBA53.htm

26/09/2007

Getting Started With the Code Composer Studio Tutorial

Page 278 sur 548



Also, the CacheTune tab of Advice Window displays text information on the conflicting symbols, such as the symbol's name, the conflict range, etc.

file:///C:/Documents and Settings/Pierre-Armand/Local Settings/Temp/~hhBA53.htm

26/09/2007

Advice Window

Below are all the symbols that have "miss" events in an area of memory that may cause `mat_xpose_oprn` to be evicted from the cache.

The "miss" events used were: LIP miss summary

A total of 9456 miss events were found.

No RTS Library functions will be checked for conflicts. Enable the RTS Library functions to include them.

Symbol	% misses that may Conflict	Address Range	Conflict Range	Section	File	Line
<code>mat_xpose_oprn</code>		0x00001000-0x000011ac		<code>.mat_xpose_oprn</code>	<code>mat_xpose_oprn.c</code>	42
<code>mat_oprn</code>	100.00%	0x00005000-0x000050e4	0x00005000-0x000050e4	<code>.mat_oprn</code>	<code>mat_oprn.c</code>	26

Welcome General CacheTune

**Note:** If the Find Conflicting Symbols button is grayed out, you can also select the particular symbol from the Mark Symbol Combo box. Once a symbol is selected, it enables the Find Conflicting Symbols button and displays the symbol name in the Address Section.

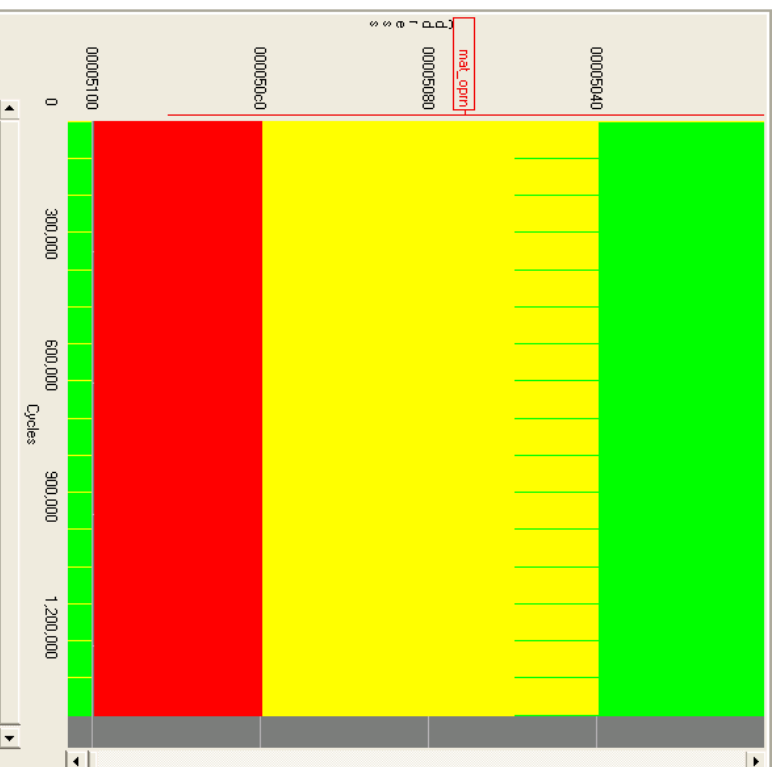
- The conflicting symbols displayed are based on memory placement, they might or might not cause cache conflict misses with selected symbol. For example, the conflicting symbols might not be referenced during the same program run. You might also want examine CacheTune graph for cache misses. Double-click on a conflicting symbol. CacheTune will zoom in to display a more detailed view of that symbol.

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007

Getting Started With the Code Composer Studio Tutorial

Page 280 sur 548

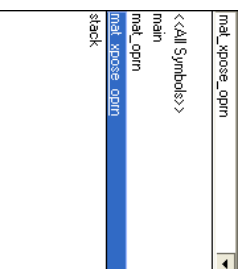



The function "mat\_xpose\_oprn" interferes in the cache with the function `mat_oprn`. When they execute back to back, they evict each other from the cache, and, subsequently, miss the next time they execute.

- Select the first empty item from the Mark Symbol Combo box. This removes all the displayed symbol names in the Address Section.

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007



4. To return to the Program Advice in Advice Window, either select the Program Cache tab, or click on the back arrow  located at the top of the Advice Window.



The Program Cache patterns of this example matches the first miss scenario discussed in the Program Cache Advice Window. The advice for eliminating the misses is to allocate the conflicting functions contiguously in memory.



### Eliminating the Cache Conflicts

#### CACHE - L3

Knowing the cache misses are conflict misses due to two functions mapped to the same cache lines, we can place the affected functions in different memory locations so that they do not overlap in cache. To achieve this efficiently, place the code of the two functions contiguously in memory.

Since the functions that execute when the interference occurs occupy less than 16KB of memory (the C64x program cache size), allocating them contiguously will eliminate any conflicts.

The two functions: `mat_xpose_oprn` and `mat_oprn` that require contiguous placement have been assigned individual sections by using the pragma `CODE_SECTION` before the definition of the functions. We can simply edit the linker command file to allocate them congruently.

1. From the Project View window, double-click on `mat_xpose_oprn_1.cmd` to open the program.

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007

### Getting Started With the Code Composer Studio Tutorial

Page 282 sur 548


2. Modify the linker command file supplied with the original source with the changes shown below. The optimized version of source file can be also found in `mat_oprn_2` directory.
 

```

MEMORY
{
    MEM0 : o = 00000000h    1 = 00001000h
    MEM1 : o = 00001000h    1 = 00004000h
    MEM2 : o = 00005000h    1 = 00004000h
    SRAM : o = 00009000h    1 = 000F7000h
    CEO : o = 80000000h    1 = 01000000h
}

SECTIONS
{
    .text:                > MEM0
    .mat_xpose_oprn       > MEM1
    .mat_oprn             > MEM1
    .stack               > MEM0
    .data                > MEM0
    .bss                 > MEM0
    .const               > MEM0
    .far                 > SRAM
    .external            > CEO
}
      
```

For details regarding the linker command file, see the *Assembly Language Tools Users' Guide (SPRU186)*.

3. Click on the Incremental Build  button. As only the linker command file is modified, this will only re-link the program.
4. Choose Debug→Reset CPU to clear the cache.
5. Choose File→Reload Program to reload the program.



### Measuring the Improvement

#### CACHE - L3

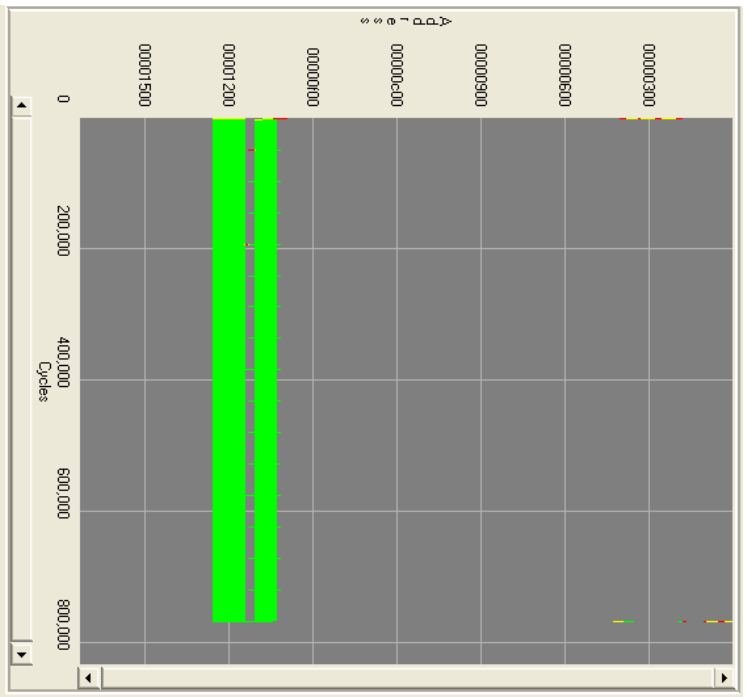
To see the result of applying this modification, repeat the steps from the first lesson.

1. From the Debug menu, select Run, and run the program to completion. Once program finishes execution, the CacheTune output window, Goals Window and Profile Viewer will update to display the new data.
2. Click on the Program Cache tab in the CacheTune window.
3. Click on the Full Zoom button view the entire trace

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007

- Roll your mouse over the LLP Misses legend above the graph to view cache misses events only. The graph should resemble the image below. Notice that red horizontal bars are eliminated and most of cache references are cache hits. The cache misses that are left are compulsory misses, and cannot be eliminated.



- Check the data in Goals Window. The data from the last execution of the program moves to the Previous column and the difference between values is displayed in the Delta column. Notice that the overall performance of the program improves 453,204 cycles after optimizing the program cache.

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007

Getting Started With the Code Composer Studio Tutorial

Page 284 sur 548

- In Profile Viewer, check the data for the LLP Stall Cycles in the Summary tab. Now the count is 106 cycles, and the percentage is 0.03%, both of which are dramatically improved. Your numbers may vary. These stalls are caused by the compulsory misses when the functions are first accessed.

This concludes the CacheTune tutorial. To learn about the full range of CacheTune capabilities and other advanced features, please see the CacheTune application report.

### Advanced Event Triggering (AET) Introduction



**AET - Intro**

This tutorial module introduces you to Advanced Event Triggering (AET) capabilities, and how they add to your ability to quickly and intuitively debug your DSP application code. AET consists of several tools, including Event Analysis and the Event Sequencer. You will examine how the different tools can tune your code and improve compiler performance.

This tutorial contains the following lessons. To go directly to a lesson, click on the link below:

[Programming Event Analysis Jobs](#)

[Programming the Event Sequencer](#)



The forward arrow will take you to the next page in this lesson.

### Programming Event Analysis Jobs



**AET - L1**

This lesson will introduce the event analysis tool and display.

Learning Objectives:

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007

- Identify the different actions of the Dynamic Menu
- Manage event analysis jobs
- Use the analysis display to program and review jobs
- Save the job configurations

Example: AET

Target Configuration:

This tutorial requires that you configure the Code Composer Studio™ IDE for a target that supports Advanced Event Triggering (AET). This tutorial was designed to run on the dsk6211 and dsk6711 targets.

**Application Objective:**

This tutorial will focus on the four key areas where tuning your C code can offer great performance improvements. A single code example, Vector Summation of Two Weighted Vectors, is used to demonstrate all four areas.

This lesson consists of the following steps. To go directly to a step, click on the link below:

[Building the Project](#)

[Programming Analysis Jobs](#)

[Managing Analysis Jobs](#)

[Programming Jobs from the Event Analysis Display](#)

[Counting Events](#)

[Saving Your Work](#)



The forward arrow will take you to the next page in this lesson.

**Building the Project**

[file:///C:/Documents and Settings/Pierre-Armand/Local Settings/Temp/~hhBA53.htm](#)

26/09/2007

**Getting Started With the Code Composer Studio Tutorial**

Page 286 sur 548

AET - L1

This tutorial will give you a quick overview of Event Analysis, and how it adds to your ability to more quickly and intuitively debug DSP application code. To prepare for this lesson, you need to build the project, load AET.out, and run to main.

1. From the Project menu, choose Open.
2. Browse to `C:\CCStudio_v3.10\tutorial\target\AET\`.
3. Select aet.pjt and click Open.
4. From the Project menu, choose Rebuild All.
5. From the File menu, choose Load Program.
6. Browse to `C:\CCStudio_v3.10\tutorial\target\ae\debug\` and select aet.out. Click Open. By default, the .out file was built into a debug directory located under your current project folder. You can change this location by selecting a different one from the main tool bar.
7. From the Debug menu, choose Go Main.

In the next lesson, you will program your first event analysis job.

**Programming Analysis Jobs**

AET - L1

After loading the Advanced Event Triggering sample project, you're ready to program an Analysis Job from the source code. The source code is where developers spend most of their time, so this is where we have enabled the most frequent debug tasks. By right-clicking in a source code window and choosing Advanced Event Triggering, you can see the Advanced Debug menu. This item expands to show several classifications of debug tasks: Breakpoints, Watchpoints, Action Points, and Counting Events.

The availability of menu items depends on the source code window selection, such as a single line, a range of lines, or a data variable.

Before you get started, you must widen the margin in main.c to view the AET icons.

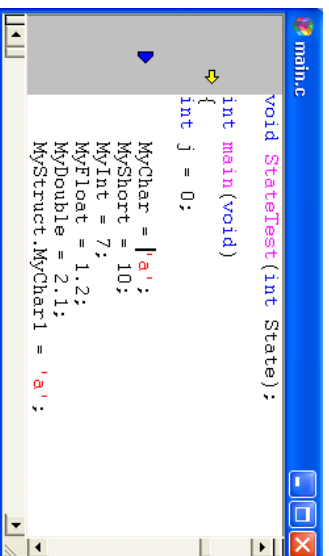
1. From the Option menu, choose Customize and select the Editor Properties tab.

[file:///C:/Documents and Settings/Pierre-Armand/Local Settings/Temp/~hhBA53.htm](#)

26/09/2007

- In the Default margin width field, change the value to 60 and click OK.
- Navigate to the main.c window in your workspace and click on line 43: that says:
 

```
MyChar = 'a';
```
- Right-click on the line you just selected and choose Advanced Event Triggering→Toggle Hardware Breakpoint. A blue icon indicates the enabled breakpoint position.



- Choose Debug→Run or click the Run icon on the toolbar to run to the breakpoint. Notice that the icon color changes to red, indicating that the breakpoint has triggered.

In a similar manner, you can create any of the other jobs in the Advanced Debug menus. For instance, a Program Action Point allows you to drive a pin to trigger a scope, or to do an RTOS Interrupt when you reach a specific code location. The actions are ISA dependent. The software identifies possible actions on each ISA or CDSP by reading a database when the system starts up.

In the next lesson, you will manage your event analysis jobs using the display.



### Managing Analysis Jobs



AET - L1

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007

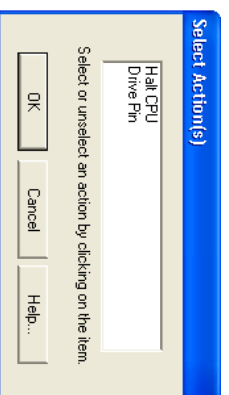
### Getting Started With the Code Composer Studio Tutorial

Page 288 sur 548

In this lesson, we'll add a watchpoint to the sample project.

- In the for loop of the main() function, select the following data variable on line 63:
 

```
MyInt = 8;
```
- Right-click on the data variable and choose Advanced Event Triggering→Toggle Action/Watch Point→Toggle Program Action Point. Notice that the menu displays new job choices with the selection of a data variable. Some jobs will only be activated on the menu if a specific range of lines is selected. The following dialog will appear:

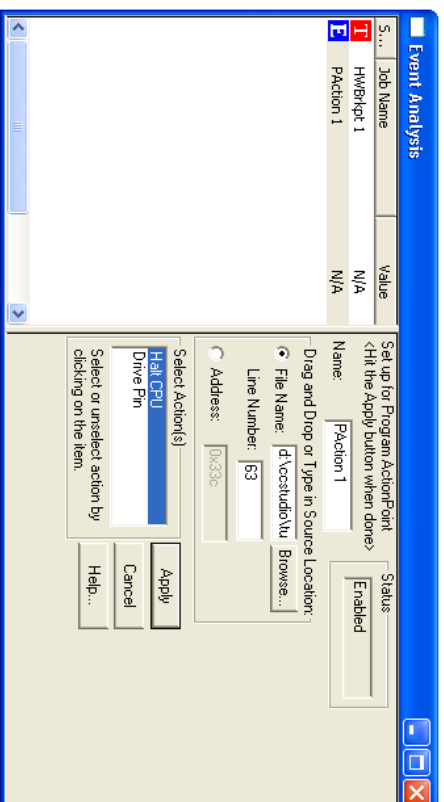


- Select Halt CPU and click OK. Main.c shows a blue icon to indicate the enabled program action point.

Now let's open the Event Analysis display. Event Analysis helps you flexibly manage jobs by allowing you to enable, disable, and remove jobs.

- From the Tools menu, choose Advanced Event Triggering→Event Analysis. This displays the Event Analysis interface.





Notice that the two previous jobs appear in the left pane of the analysis display.

The breakpoint we set (HWBrkpt 1) displays a red icon to indicate this job has been triggered by running the program after setting the breakpoint in [Programming Analysis Jobs](#). The program action point (Paction 1) displays a blue icon to indicate this job is enabled.

Click on the first job (HWBrkpt 1) in the Event Analysis display to select it. This displays the setup dialog for that job in the right hand pane of the display.

Now you can edit the properties for that job in the Event Analysis window. You may enable, disable, or remove Event Analysis jobs by right clicking on the job in the Analysis Display and selecting a choice from the menu.

In the next lesson, you will program a job from the event analysis display.



### **Programming Jobs From the Event Analysis Display**



AET - L1

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007

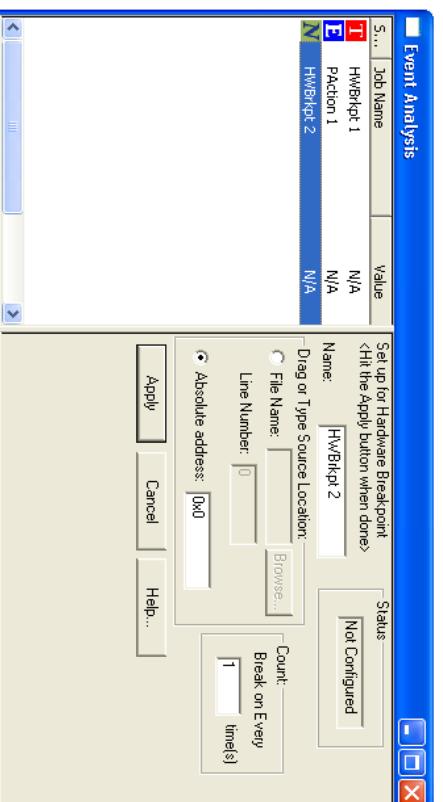
### **Getting Started With the Code Composer Studio Tutorial**

Page 290 sur 548

In the first part of this lesson, you learned how to create an Event Analysis job by right clicking in the source code. You then used the Event Analysis Display to view and manage your job. You can also create jobs directly from the Event Analysis display window using the right click menu and drag and drop actions. You can drag and drop lines of code or data variables from the source window into the Event Analysis Display. The source location and target configuration determine which jobs will be created.

This time, you'll program a breakpoint from within the analysis display by dragging a line of source code from main.c.

1. Right click in the Event Analysis display and choose **Set Hardware Breakpoint**. If necessary, enlarge the display so you can see the Set up for Hardware Breakpoint dialog. Notice that the program has created a default name and count of 1 for your breakpoint.



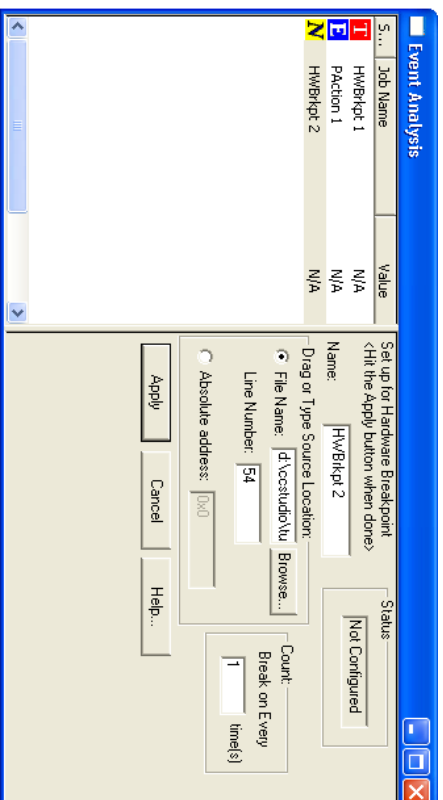
2. In main.c, select the following line (line 54) of code using the left mouse button:

```
MyStruct.MyFloat = 1.12;
```

3. Use the left mouse button again to drag your selection into the Set up for Hardware Breakpoint dialog. This action adds the file name, path, and line number to the dialog, and adds a yellow icon in the margin of main.c, next to the line of code.

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007



- If the information in the dialog is correct, click Apply to program the Event Analysis job. Notice that the yellow icon changes to blue in both the Event Analysis Display and the source file to indicate the job is now enabled.

You could also easily drag a data variable from the source code window to set up a breakpoint.

**Note:** Only the Event Analysis display dialog for a breakpoint includes a procedure to specify a count. Setting a breakpoint by right clicking in the source code window does not include the count specification because we wanted to streamline the menu selection.

In the next lesson, you will learn how to count events for debugging purposes.



## Counting Events



file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007

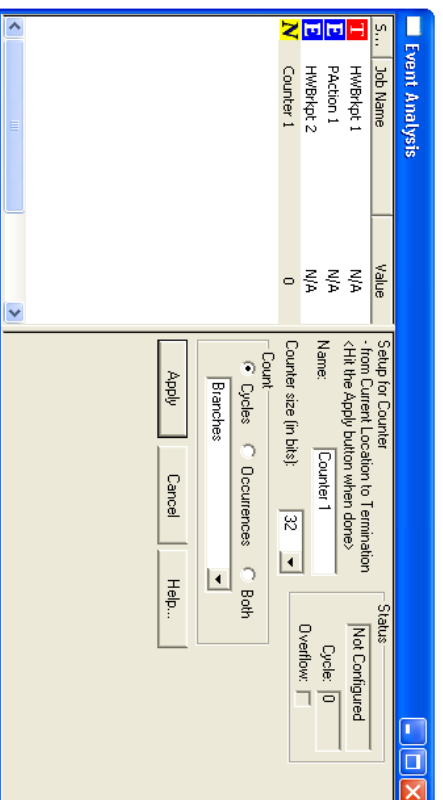
Getting Started With the Code Composer Studio Tutorial

Page 292 sur 548

AET - L1

Breakpoints and watch points are frequently-used debug operations, along with the ability to count events.

- Right click in the Event Analysis display and choose Set Counter (from Current Location).



The count dialog has many options. In order to successfully program a count job, you must specify the following:

- In the Counter Size field, select 32 from the drop down dialog box.
- In the Count field, select an event to count from the drop down dialog box.
- Indicate whether you want to count occurrences of that event, the number of cycles spent in that event, or both by enabling one of the three radio buttons.
- Click Apply to enable the job.
- You may get a message about conflicting analysis jobs. Select a job to disable, and click Disable Selection.

The Analysis Display will show a grey icon next to the disabled job in the Event Analysis display, and a black icon next any disabled line of source code in main.c.

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007

In the next lesson, you will save your workspace, so that these jobs will be available.



### Saving Your Work

**AET - L1**

Now that you have programmed several Event Analysis jobs from the source code and from the Event Analysis display, you may want to save those jobs for your next debug session. Event Analysis jobs are saved as part of your Code Composer Studio™ workspace. To save your workspace and the Event Analysis jobs associated with it:

1. From the File menu, choose Workspace→Save Workspace As.
2. Browse to [C:\CCStudio\\_v3.10\tutorial\target\ael\](#) and type `aet` in the file name field and click Save to save your workspace and the associated Event Analysis jobs.

To prepare for the next lesson, [Programming the Event Sequencer](#), you must remove all the jobs and hide the Event Analysis display.

3. Right click in the Event Analysis display and choose Disable All Jobs.
4. Click Yes to accept this action.
5. Right click in the Event Analysis display and choose Hide.
6. From the File menu, choose Workspace→Save Workspace.



### Programming the Event Sequencer

**AET - L2**

This tutorial will give you a quick overview of the Event Sequencer, and how it adds to your ability to quickly and intuitively debug DSP application code.

Learning Objectives:

[file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm](#)

26/09/2007

### Getting Started With the Code Composer Studio Tutorial

Page 294 sur 548

- Create an Event Sequencer program
- Describe states
- Define an JF statement
- Determine the appropriate event or action
- Run the program
- Save your work

Example: AET

Target Configuration:

This tutorial requires that you configure the Code Composer Studio™ IDE for a target that supports Advanced Event Triggering (AET). This tutorial was designed to run on the dsk6211 and dsk6711 targets.

Application Objective:

The example code, `main.c`, modifies element zero many times between the call to `InitSideA()` and `InitSideB()`. For this lesson, you will use the Event Sequencer to create a program that will wait in a state (State 1) until `InitSideA()` has finished, then move on to the next state (State 2) and perform a simple watchdog.

This lesson consists of the following steps. To go directly to a step, click on the link below:

[Building the Project](#)

[Creating a State](#)

[Completing Your Program](#)

[Saving Your Work](#)

[Tips and FAQs](#)



### Building the Project



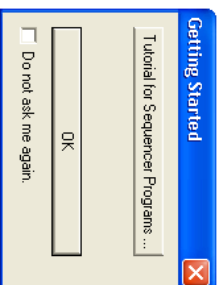
[file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm](#)

26/09/2007

Your first Event Sequencer program will detect a memory corruption bug in the sample project. After the function, `IntSideA()`, is called, the first element of the array, `SideA`, is corrupted. The call to the function, `IntSideB()`, is the culprit, but you will use AET to discover this.

To prepare for this lesson, you will build the sample project, load AET.out, and run to main.

1. From the Project menu, choose Open, or open the workspace you saved in the previous lesson.
2. Browse to `C:\CCStudio_v3.10\tutorial\target\AET\`.
3. Select `set.pjt` and click Open.
4. From the Project menu, choose Rebuild All.
5. From the File menu, choose Load Program.
6. Browse to `C:\CCStudio_v3.10\tutorial\target\set\debug\`. Select `set.out` and click Open. By default, `set.out` file was built into a debug directory located under your current project folder (AET). You can change this location by selecting a different one from the main tool bar.
7. From the Debug menu, choose Go Main. If you are using the same workspace you used in the Event Analysis lesson, you should see black icons indicating any disabled jobs.
8. From the Tools menu, choose Advanced Event Triggering→Event Sequencer.



**Note:** The first time you use the Event Sequencer, you will have an option to select a Tutorial before the tool launches. Click OK to launch the Event Sequencer.



### Creating a State

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007

Getting Started With the Code Composer Studio Tutorial

Page 296 sur 548



AET - L2

If you are using a 6711DSK target, click [here](#).

If you are using a 6211DSK target, click [here](#).



### Completing Your Program



AET - L2

If you are using a 6711DSK target, click [here](#).

If you are using a 6211DSK target, click [here](#).



### Saving Your Work



AET - L2

You can save Event Sequencer programs for later use by right clicking in the Event Sequencer display and choosing Save Sequencer Program. The Sequencer program is saved with a `.seq` file extension. You can also save Sequencer programs by saving your workspace. Saving your workspace does not create a `.seq` file, but the workspace file will remember your work environment setup.

1. Right click in the Event Sequencer display and choose Save Sequencer Program.
2. In the File Name field, type *first*.
3. Click Save to save your program.

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007

**Note:** If you want to save your Sequencer program as part of your workspace: From the File menu, choose Workspace → Save Workspace.

Once you have created a sequencer program, you can disable or re-enable it as needed during the execution of your program. To disable or enable your Sequencer program:

Right-click in the Event Sequencer window and choose Disable Sequencer Program, or Enable Sequencer Program.

This tutorial does not present all of the capabilities of the Event Sequencer. You can read more about Advanced Event Triggering by choosing Help→Contents and clicking on the Debugging→Analysis Tools for Debugging→Advanced Event Triggering book. Here are some items you can explore on your own by using the right-click menu in the Event Sequencer:

System Events

Bus Events

Counting Events

Global Actions

Global If Statements



## Tips and FAQs



AET - L2

1. The AET tool menu is created dynamically based on your target configuration.
2. States are made up of If statements.
3. If statements are made up of Undefined Events and Undefined Actions
4. Undefined Events are of four types: Data Event, Program Event, System Event, and Bus Event.
5. Undefined Actions are defined from the Action list in the Sequencer Display or by choosing Define Action from the right-click menu.
6. State level Actions are added to a State, in addition to the If statement, using the right-click menu and are only available when the State is selected.
7. Global If statements are not part of a State, and are different from the If statements found in States.

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007

Getting Started With the Code Composer Studio Tutorial

Page 298 sur 548

8. Global Actions are not part of a State, and are different than Undefined Actions.
9. Together, all of these items make your Event Sequencer program.

### Tips

Setting Hardware Breakpoints with AET is easy. When you are configured for a target that has AET support, double-click in source code located in ROM and CCS uses AET to set a Hardware Breakpoint instead of a Software Breakpoint.

When you create Hardware Breakpoints in the CCS Breakpoint dialog (Debug→Breakpoints), you can see the Breakpoint in the Analysis display.

This concludes the Advanced Event Triggering tutorial.



## Optimizing C Introduction

OPTC - Intro

This lesson walks you through the code development flow and introduces you to compiler optimization techniques. It uses step-by-step instructions and code examples to show you how to use the software development tools in each phase of development.

[More about the Compiler](#)

Learning Objectives:

- Identify the different aspects of the project environment
- Prepare the project for the compiler by passing the necessary information
- Improve compiler performance using build options

Example: optimizing\_C

Target Configuration:

This tutorial was created with the 6211 CPU Cycle Accurate Simulator, little endian. It is possible to use other 62xx and 64xx simulators, but the output will differ.

Application Objective:

This tutorial will focus on the four key areas where tuning your C code can offer great performance improvements. A single code example, Vector Summation of Two Weighted Vectors, demonstrates all four areas.

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007

This lesson consists of the following steps. To go directly to a step, click on the link below:

- [Project Familiarization](#)
- [Loop Carry Path From Memory Pointers](#)
- [Balancing Resources With Dual-Data Paths](#)
- [Packed Data Optimization of Memory Bandwidth](#)
- [Summary](#)
- [Writing Linear Assembly](#)



The forward arrow will take you to the next page in this lesson.

## Project Familiarization



### OPTC - Familiarization

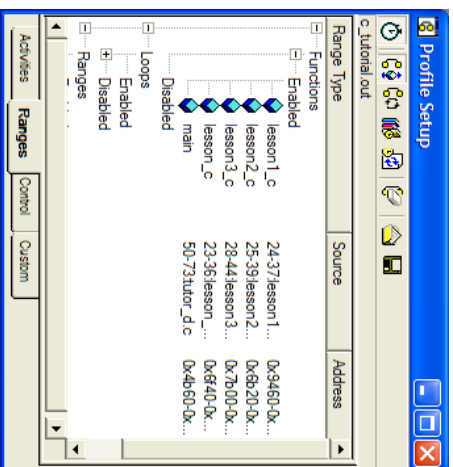
For your convenience, this project was built and saved as a Code Composer Studio Project (\*.pj). For the best results, run this tutorial on the C62xx CPU Cycle Accurate little endian target. To get started, you will load the project and take a look at the source files by following these steps:


1. From the Project menu, choose Open, and browse to: C:\CCStudio\_v3.10\tutorial\target\optimizing\_c\
  2. Select c\_tutorial.pjt, and click Open to load the project.
  3. From the Project menu, choose Rebuild All.
  4. From the File menu, choose Load Program, and browse to: C:\CCStudio\_v3.10\tutorial\target\optimizing\_c\Debug\
  5. Select c\_tutorial.out, and click Open to load the file.
- The disassembly window with a pointer at c\_int00 is displayed.

6. In the Project View window, right-click on tutor\_d.c and select Open. Scroll down to the main function.

Next, you will setup for profiling so you can get a quick look at the source files for this tutorial:

1. From the Profile menu, choose Setup.
2. Check the icons at the top of the Profile Setup tool to determine if profiling is enabled. The Enable/Disable Profiling icon  must be toggled on to enable profiling. Profiling may not be enabled if the program has started running or if it was explicitly disabled. Data will not be collected if profiling is not enabled.
3. Under the Ranges tab, click the Enable/Disable All Functions Icon  to enable function profiling. The Profile window automatically sets up profiling for each function in your project. The Profile Setup window should now look like this:



4. From the Custom tab, select the event Cycle CPU. This will count the cumulative total of CPU cycles for all functions, ignoring system effects.
5. From the Profile menu, choose Viewer.
6. From the Debug menu, choose Run. You can also click on the Run icon , or F5 key to run the program. This updates the Profile Viewer window statistics and Disassembly window.
7. Double-click on the header of column cycle:CPU: Excl. Total to sort the functions in descending order. These are sample numbers, your numbers may vary.

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007

Getting Started With the Code Composer Studio Tutorial

Page 302 sur 548

Address Range	Symbol Name	SLR	Symbol Type	Access Count	cycle:CPU: Incl. ...	cycle:CPU: Excl. ...
0:0x49760-0x998c	main	50-73:tutor_d.c	function	1	2610	1503
0:0xae6c0-0xb020	lesson_c	23-36:lesson1_c.c	function	1	103	103
0:0xbdb00-0xbef00	lesson1_c	24-37:lesson1_c.c	function	1	97	97
0:0xac000-0xad5c	lesson2_c	25-39:lesson2_c.c	function	1	69	69
0:0xb160-0xb28c	lesson3_c	28-44:lesson3_c.c	function	1	49	49

While running on the 6211 CPU Cycle Accurate Simulator, you can see cycle:CPU counts in the Incl. Total column of 103, 97, 69, and 49 for the functions in c\_tutorial.out. Each of these functions contains the same C code but has some minor differences related to the amount of information to which the compiler has access.

The rest of this tutorial discusses these differences and teaches you how and when you can tune the compiler to obtain performance results comparable to fully optimized hand-coded assembly.

Now you are ready to start the Loop Carry Path From Memory Pointers portion of the lesson.



**Loop Carry Path From Memory Pointers**

OPTC - L1

When you rebuild the project in **Project Familiarization**, each file was compiled with compiler options -k -q -o3 -fr \*. \Debug" -mhw -mv6200. Because it used the -k option, \*asm file for each included \*.c file. Here, you will examine the c source and the resulting assembly output to check for potential improvements.

1. In the Project View window, right-click on lesson\_c.c and select Open.
2. From the File menu, choose Open.
3. From the Files of Type drop-down menu, choose Asm Source Files (\*.a\*, \*.s\*).

[lesson\\_c.c](#)

2. From the File menu, choose Open.

3. From the Files of Type drop-down menu, choose Asm Source Files (\*.a\*, \*.s\*).

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007

4. Select lesson\_c.asm and click Open.

Each .asm file contains software pipelining information. You can scroll down to see the feedback from lesson\_c.c, in lesson\_c.asm:

[lesson\\_c.asm](#)

The loop has been duplicated. One is scheduled with  $ii = 10$ , implying that each iteration of the loop takes ten cycles. The other is scheduled at  $ii = 2$ .

**Question:** Why is the loop duplicated?

**Answer:**

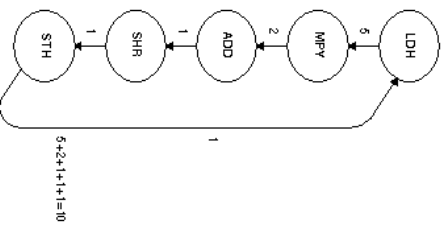
**Question:** Where are the problems with this loop?

**Answer:**

**Question:** Why did the loop start searching for a software pipeline at  $ii=10$  (for a 10-cycle loop)?

**Answer:**

You can also use a dependency graph to analyze feedback, for example:



file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007

Getting Started With the Code Composer Studio Tutorial

Page 304 sur 548

**Question:** Why is there a dependency between STH and LDH? They do not use any common registers, so how can there be a dependency?

**Answer:**

**Question:** Is there any dependency between xptr, yptr, and w\_sum?

**Answer:**

The main calling function in tutor\_d.c indicates that these pointers all point to separate arrays in memory. However, this information is not available from the compiler's local view of lesson\_c.

**Question:** How can you pass more information to the compiler to improve its performance?

**Answer:**



**Adding a Restrict Qualifier**

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007



In this portion of the lesson, you will add a restrict type qualifier to lesson\_c.c to pass more information to the compiler and to improve loop performance. Lesson\_c already has the change, and the results can be reviewed in Lesson1\_c.asm.

1. In the Project View window, open the Source folder, right-click on lesson1\_c.c and select Open.
2. From the File menu, select Open.
3. From the Files of Type drop-down menu, select Asm Source Files (\*.a\*, \*.s\*)
4. Select lesson1\_c.asm and click Open.

#### [lesson1\\_c.c](#)

The only change made in lesson1\_c is the addition of the restrict type qualifier for xptr and yptr. Since we know that these are actually separate arrays in memory from w\_sum, in function lesson1\_c, we can declare that nothing else points to these objects. No other pointer in lesson1\_c.c points to xptr and no other pointer in lesson1\_c.c points to yptr. Because of this declaration, the compiler knows that there is no possible dependency between xptr, yptr, and w\_sum. Compiling this file creates feedback as shown in lesson1\_c.asm, a shortened version is available here:

#### [lesson1\\_c.asm](#)

Now, the Loop Carried Dependency Bound is one. By simply passing more information to the compiler, we allowed it to improve a 10-cycle loop to a 2-cycle loop. Meanwhile, the duplicated loop is eliminated and this reduces the code size.

**Tip: Use Program Level Optimization.** Later in the tutorial, you learn how the compiler retrieves this type of information automatically by gaining full view of the entire program with program level optimization switches. See the [TMS320C6000 Optimizing Compiler User's Guide \(SPRU187\)](#) for more information on the restrict type qualifier.



### Solving Alias Disambiguation Problems

A special option in the compiler, -mt, tells the compiler to ignore alias disambiguation problems like the one described in lesson\_c. Try using the -mt option to rebuild lesson\_c.c and look at the results.

1. From the Project menu, choose Build Options to view the Build Options Dialog.
2. Select the Compiler tab.

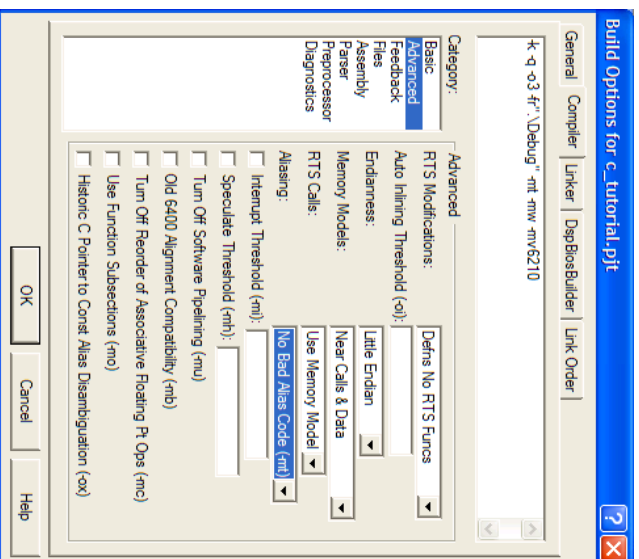
file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007

### Getting Started With the Code Composer Studio Tutorial

Page 306 sur 548

3. In the Category box, select Advanced.
4. In the Aliasing drop-down box, select No Bad Alias Code to enable the -mt option.



5. Click OK to set the new options.
6. Open lesson\_c.c by selecting it in the project environment, or double-clicking on it in the Project View window.
7. From the Project menu, choose Build, or click on the Build icon.
8. If prompted, reload lesson\_c.asm and lesson1\_c.asm.

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007

9. From the File menu, choose Open, and select lesson\_1.asm.

In the editor window for lesson\_1.asm, you see that the file header contains a description of the options used to compile the file under Global File Parameters. The following line shows that was used:

```
/* Memory Aliases : Preamble not aliases (optimistic)
```

Scroll down until you see the feedback embedded in lesson\_1.asm.

You now see the following:

```
/* Loop Carried Dependency Bound(*) : 0
* ii = 2 Schedule found with 5 iterations in parallel
```

This indicates that a 2-cycle loop was found. [Balancing Resources With Dual Data Paths](#) will address information about potential improvements to this loop.



## Summary

**OPTC - L1**

Status Update: Compare lesson\_1.c to lesson\_1.c.

Consideration	Lesson_1_C	Example Lesson1_C
Potential Pointer Aliasing Information (discussed in Loop Carry Path From Memory Pointers)	X	✓
Loop Count Information (minimum trip count)	X	X
Loop Count Information (maximum trip count)	X	X
Alignment Information - xptr and yptr aligned on a word boundary	X	X

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007

## Getting Started With the Code Composer Studio Tutorial

Page 308 sur 548

Cycles per Iteration

10

2



## Balancing Resources With Dual-Data Paths

**OPTC - L2**

The previous exercise showed you a simple way to make large performance gains in lesson\_1.c. The result is lesson\_1.c with a 2-cycle loop.

**Question:** Is this the best the compiler can do? Is this the best possible result on the Velocity architecture?

**Answer**

1. From the File menu, choose Open.
2. From the Files of Type drop-down menu, select Asm Source Files (\*.a\*, \*.s\*).
3. Select lesson\_1.asm and click Open.

A shortened example follows here:

[lesson\\_1.asm](#)

The first iteration interval (ii) attempted was two cycles because the Partitioned Resource Bound is two. Looking at the .D units and the .T address paths will show us the reason for the cycles. This loop requires two loads (from xptr and yptr) and one store (to w\_sum) for each iteration of the loop.

Each memory access requires a .D unit for address calculation, and a .T address path to send the address out to memory. Because the C6000 has two .D units and two .T address paths available on any given cycle (A side and B side), the compiler must partition at least two of the operations on one side (the A side). These operations are the bottleneck in resources (highlighted with an **h**), and are the limiting factor in the Partitioned Resource Bound. The feedback in lesson\_1.asm indicates an imbalance in resources between the A and B side due, in this case, to an odd number of operations being mapped to two sides of the machine.

**Q uestion:** Is it possible to improve the balance of resources?

**Answer**

**Question:** Why didn't the compiler unroll the loop?

**Answer**

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007



## Passing Loop Count Information

OPTC - L2

In this exercise, loop count information will be passed to the compiler using the `MUST_ITERATE` pragma, as demonstrated in lesson2.c.c.

In the Project View window, double-click on Lesson2\_c.c to open it.

[lesson2\\_c.c](#)

In lesson2\_c.c, no code is altered, only additional information is passed via the `MUST_ITERATE` pragma. We simply guarantee to the compiler that the trip count (N, in this case) is a multiple of two and that the trip count is greater than or equal to 10.

The first argument for `MUST_ITERATE` is the minimum number of times the loop will iterate. The second argument is the maximum number of times the loop will iterate. The trip count must be evenly divisible by the third argument. For this example, we chose a trip count large enough to tell the compiler that it is more efficient to unroll. Always specify the largest safe minimum trip count. Now let's open lesson2\_c.asm and examine the feedback.

1. From the File menu, choose Open.
2. From the Files of TType drop-down menu, select Asm Source Files (\*.a\*, \*.s\*).
3. Select lesson2\_c.asm and click Open.

[lesson2\\_c.asm](#)

Notice the following things in the feedback:

A schedule with three cycles (i1=3): Looking at the .D units and .T address paths will tell you that this 3-cycle loop comes after the loop has been unrolled because the resources show a total of six memory accesses evenly balanced between the A side and B side. Therefore, our new effective loop iteration interval is 3/2 or 1.5 cycles.

A Known Minimum Trip Count of 10: This is because we specified the count of the original loop to be greater than ten and a multiple of two (at least twenty-two). Unrolling cuts this in half.

Therefore, by passing information without modifying the loop code, compiler performance improves from a 10-cycle loop to 2 cycles, and now to 1.5 cycles.

**Question** Is this the lower limit?

**Answer**

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007

## Getting Started With the Code Composer Studio Tutorial

Page 310 sur 548



## Summary

OPTC - L2

Status Update: Compare lesson\_c to lesson1\_c and lesson2\_c.

Consideration	Lesson_c.c	Lesson1_c.c	Lesson2_c.c
Potential Pointer Aliasing Information (discussed in Loop Carry Path From Memory Pointers)	X	✓	✓
Loop Count Information (minimum trip count)	X	X	✓
Loop Count Information (maximum trip count)		X	✓
Alignment Information - xptr and yptr aligned on a word boundary	X	X	X
Cycles per Iteration	10	2	1.5

## Packed Data Optimization of Memory Bandwidth

OPTC - L3

**Balancing Resources With Dual-Data Paths** produced a 3-cycle loop in lesson2\_c.c that performed two iterations of the original vector sum of two weighted vectors. This means that each iteration of our loop now performs six memory accesses, four multiplies, two adds, two shift operations, a decrement for the loop counter, and a branch. You can see this phenomenon in the feedback of lesson2\_c.asm.

1. From the File menu, select Open.
2. From the Files of TType drop-down menu, select Asm Source Files (\*.a\*, \*.s\*).
3. Select lesson2\_c.asm and click Open.

[lesson2\\_c.asm](#)

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007

Notice the following things in the feedback:

The six memory accesses appear as .D and .T units. The four multiplies appear as .M units. The two shifts and the branch show up as .S units. The decrement and the two adds appear as .LS and .LSD units. Due to partitioning, they don't all show up as .LSD operations. Two of the adds must read one value from the opposite side. Because this operation cannot be performed on the .D unit, the two adds are listed as .LS operations.

By analyzing this part of the feedback, we can see that the memory accesses limit the resources the most; hence, the reason for an asterisk highlighting the .D units and .T address paths.

**Question:** Does this mean that we cannot make the loop operate any faster?

**Answer:**

In the unrolled loop generated from lesson2\_c, we load two consecutive 16-bit elements with LDHs from both the xptr and yptr array.

**Question:** Why not use a single LDW to load one 32-bit element, with the resulting register load containing the first element in one-half of the 32-bit register and the second element in the other half?

**Answer:**

**Question:** Why doesn't the compiler do this automatically in lesson2\_c?

**Answer:**



### Aligning Pointers With `_nassert`

OPTC - L3

In order to perform a LDW (32-bit load) on the C63x, C67x, and C64x cores, the address must be aligned to a word address; otherwise, it loads incorrect data. An address is word-aligned if the lower two bits of the address are zero. Unfortunately, in our example, the pointers, xptr and yptr pass into lesson2\_c.c with no local scope knowledge of their values. Therefore, the compiler must act conservatively and assume that these pointers might not be aligned. Once again, we can pass more information to the compiler, this time via the `_nassert` statement. Lesson3\_c.c demonstrates this technique.

**Note:** When writing C++ code, be aware that `_nassert` is in the standard namespace. This means that when including the `cassert` header file, the correct syntax is: `std::_nassert (expr::easi.on)`

1. In the Project View window, right-click on lesson3\_c.c and select Open.

[lesson3\\_c.c](#)

By asserting that xptr and yptr addresses "anded" with 0x3 are equal to zero, the compiler knows that they are word aligned. This means the compiler can perform LDW and packed data optimization on these memory accesses.

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007

### Getting Started With the Code Composer Studio Tutorial

Page 312 sur 548

2. From the File menu, select Open.
3. From the Files of Type drop-down menu, select Asm Source Files (\*.a\*, \*.s\*).
4. Select lesson3\_c.asm and click Open.

[lesson3\\_c.asm](#)

Success! The compiler has fully optimized this loop. You can now achieve two iterations of the loop every two cycles for one cycle per iteration throughout.

The .D and .T resources now show four (two LDWs and two STHs for two iterations of the loop).



### Summary

OPTC - L3

Status Update: Compare lesson\_c to lesson1\_c, lesson2\_c, and lesson3\_c.

Consideration	Example			
	Lesson_c.c	Lesson1_c.c	Lesson2_c.c	Lesson3_c.c
Potential Pointer Aliasing Information (discussed in Loop Carry Path From Memory Pointers)	X	✓	✓	✓
Loop Count Information (minimum trip count)	X	X	✓	✓
Loop Count Information (maximum trip count)	X	X	✓	✓
Alignment Information - xptr and yptr aligned on a word boundary	X	X	X	✓
Cycles per Iteration	10	2	1.5	1



### Writing Linear Assembly

OPTC - L4

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007

When the compiler does not fully exploit the potential of the C6000 architecture, you may increase performance quality by writing your loop in linear assembly. Linear assembly is the input for the assembly optimizer.

Linear assembly resembles regular C6000 assembly code in that you use C6000 instructions to write your code. With linear assembly, however, you do not need to specify all of the information that you need to specify in regular C6000 assembly code. With linear assembly code, you have the option of specifying the information or letting the assembly optimizer specify it for you. Here is the information that you do not need to specify in linear assembly code:

- Parallel instructions
- Pipeline latency
- Register usage
- Which functional unit is being used

If you choose not to specify these things, the assembly optimizer determines the information that you did not include, based on the your code information. As with other code generation tools, you might need to modify your linear assembly code until you are satisfied with its performance. When you do this, you will probably want to add more detail to your linear assembly. For example, you might want to specify which functional unit should be used.

Before you use the assembly optimizer, you need to know the following things about how it works:

- A linear assembly file must be specified with a .sa extension.
- Linear assembly code should include the .cproc and .endproc directives. The .cproc and .endproc directives delimit a section of your code for optimization. Use .cproc at the beginning of the section and .endproc at the end of the section. Thus, you can specify sections of your assembly code for optimization, like procedures or functions.
- Linear assembly code may include a .reg directive. The .reg directive allows you to use descriptive names for stored register values. When you use .reg, the assembly optimizer chooses a register whose use agrees with the functional units chosen for the instructions that operate on the value.
- Linear assembly code may include a .trip directive. The .trip directive specifies the value of the trip count. The trip count indicates how many times a loop will iterate.

Let's look at a new example, `lirfat`, which will show the benefit of using linear assembly. The compiler does not optimally schedule this loop. Thus, the `lircas4` function does not improve with the C modification techniques from first lessons of the tutorial. In order to get the best partition, we must write the function in partitioned linear assembly.

In order to follow this example, you must open the project:

1. From the Project menu, choose Open.
2. Browse to `C:\CCStudio_v3.10\tutorial\target\linear_asm\` and select `tutorial.pjt`
3. From the Project menu, choose Rebuild All.
4. In the Project View window, right-click on `lirfat_L1.c` and select Open.

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007

## Getting Started With the Code Composer Studio Tutorial

Page 314 sur 548

[lirfat Function in C](#)

5. Next, look at the c source file and the software pipeline information feedback in the generated assembly files.
6. From the File menu, choose Open.
7. Open the Debug folder.
8. From the Files of TYPE drop-down menu, choose Asm Source Files (\*.a\*, \*.s\*).
9. Select `lirfat_L1.asm` and click Open.

The following example shows a condensed assembly output from the `lirfat` Function in the C example above.

[Software Pipelining Feedback From the lirfat C Code](#)

From the feedback in the generated in the .asm file, we can see that the compiler generated a sub-optimal loop: it can not find a schedule at iteration interval (ii) of 2 because of register live-too-long problem.

Notice that one register must be live for at least 3 cycles. As no value can be live longer than the minimum iteration interval (ii), the minimum ii must be 3 cycles. One solution to the live-too-long problem is to break up the lifetime of the register by inserting move (MV) instruction. This would allow us to schedule at an iteration interval (ii) of 2 instead of the current ii of 3. For more information of the live-too-long issues and the methods used to solve them, please refer to Chapter 6 of *TMS320C6000 Programmer's Guide (SPRU199)*.



## Re-Writing the Example

OPTC - L4

When the Partitioned Resource Bound is higher, this usually means we can make a better partition by writing the code in linear assembly. The following example shows how to rewrite the `lirfat ()` function using linear assembly.

### Rewriting the `lirfat ()` Function in Linear Assembly

```
.include "lirfat_L1.h62"
.no_mdep
;
.reg A_xT, A_x, A_D, B_K, A_BK, B_rtk, A_L, B_DI, A_xT, B_J
.mptc A_DP, B_D, B_D, B_D, B_D, B_K, B_DP
.mptc A_DP, x, 4
.mptc B_DP, x+4, 4
.mptc K_DP, y, 2
MV B_rtk, B_rtk, A_rtk
MV A_rtk, A_rtk
```

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007

```

loop_j:
LDH      *_A_xp++,   A_x      ; x[j]
SHL      A_x,      16,      A_xt ; xt=x[i] j<<16
MV       B_nk,     A_1      ; i=nk
ADDAH   A_kp,     A_1      A_nk ; &k[i]k
ADDAW   A_dp,     A_dp,    A_nk, A_dp ; &b[i]k
MV       A_dp,    B_dp

loop_i:
LDW     *--A_dp,   A_b      ; b[i]
LDH     A_b,     *--A_kp,  B_k   ; k[i]
SMPYHH SMPYHH   A_b,     B_k,   A_bk ; bk=b[i]*k[i]
SUB     A_xt,    A_bk,    A_rt   ; rt=bk
MPY     B_k,     A_bk,    A_rt   ; rt=bdk
SMPYHH SMPYHH   A_xt,    B_k,   B_k ; rtk=rt*k[i]
MV      A_b,     A_b,     B_b
MV      B_b,     B_b,     B_b
MV      B_b,     B_b,     B_b
ADD     B_d,     B_d,     B_b
ADD     B_d,     B_d,     B_b
STW     A_1,    SUB     A_1,    loop_i ; b1=b[i]+rtk
STW     A_xt,   A_xt,    16,      A_x ; b1=b[i]=b1
SHR     A_x,    *B_xp++ ; xt>16
STH     A_x,    B_j,     B_j ; x[j]=x
SUB     B_j,    B_j,     loop_j
[B_j]
[B_j]
B
.endproc

```

The following example shows the software pipeline feedback from the `lirfat ()` Function in linear assembly above.

#### [Software Pipeline Feedback from Linear Assembly](#)

Notice in the assembly code that a `move (MV)` instruction is now manually inserted to reduce the lifetime of the register to 2. From the software pipeline feedback information in the previous example, you can see that a software pipeline schedule is found at `li = 2`. This is a result of rewriting the `lirfat ()` function in linear assembly, as shown above.

This concludes the Optimizing C tutorial.



### [Using This Tutorial Module](#)

#### [DSP/BIOS - Intro](#)

The DSP/BIOS tutorial module introduces the basics of [DSP/BIOS](#) and provides additional lessons about specific modules of the DSP/BIOS API.

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007

### Getting Started With the Code Composer Studio Tutorial

Page 316 sur 548

#### [Prerequisites for the DSP/BIOS Tutorial Module](#)

Before using the DSP/BIOS tutorial module, you should have done the following:

- Installed Code Composer Studio. Except where otherwise noted, this tutorial can be used with any supported board or a simulator. These profiling lessons were tested with a 62xx CPU Cycle Accurate Device Simulator. Using different configurations may have unexpected results.
- Learned how to use Code Composer Studio to create, build, run, and test a program. For example, you should be able to set breaks and step through a program. To learn the basics of using Code Composer Studio, see the section of this tutorial on the [Code Composer Studio IDE](#).

#### [Using the DSP/BIOS Tutorial Module](#)

Begin this tutorial by performing the introductory lessons in the sequence provided. This gives you a basic overview of DSP/BIOS.

After the introductory lessons, you can perform the remaining lessons in any sequence. You don't need to complete all the lessons. Just use lessons as you encounter DSP/BIOS concepts and modules you want to understand.

**Target Configuration:** These profiling lessons were tested with a 62xx CPU Cycle Accurate Device Simulator. Using different configurations may have unexpected results. The graphs and figures used in the tutorial reflect the 62xx CPU Cycle Accurate Simulator; your results may differ depending on your device. For more information on using Setup to choose different configurations in Code Composer Studio 3.0, please see the [Configuring Target Devices](#) tutorial. Also, certain profiling options may not appear using a 54x or 28x target.

You can learn more about DSP/BIOS from the DSP/BIOS manuals and the online help system that opens when you press F1 or click Help within Code Composer Studio.

This tutorial module contains the following lessons:

[Getting Started With DSP/BIOS](#)

[Scheduling Program Threads](#)

[Sharing and Synchronizing Resource Access](#)

[Managing Memory](#)

[Handling Input/Output](#)



### [What is DSP/BIOS?](#)

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007

DSP/BIOS is a scalable real-time kernel. It is designed for applications that require real-time scheduling and synchronization, host-to-target communication, along with real-time instrumentation. DSP/BIOS provides preemptive multi-threading, hardware abstraction, and real-time analysis.

DSP/BIOS is packaged as a *set of modules* that can be linked into an application; applications include only those functions of the DSP/BIOS that are referenced (directly or indirectly) by the application. In addition, the DSP/BIOS Configuration Tool allows you to optimize code size and speed by disabling DSP/BIOS features not used in their programs.

You can use DSP/BIOS to instrument any application to be probed, traced, and monitored in real-time. Programs that use the DSP/BIOS Configuration Tool to take advantage of the multi-threading capabilities of DSP/BIOS are implicitly instrumented.

DSP/BIOS is integrated with Code Composer Studio, requires no runtime license fees, and is fully supported by Texas Instruments™. DSP/BIOS is a key component of TI's eXpressDSP™ technology.

DSP/BIOS includes the following components:

- DSP/BIOS Configuration Tool: This tool allows you to create and configure the DSP/BIOS objects used by your program. You can also use this tool to configure memory, thread priorities, and interrupt handlers.
- DSP/BIOS Real-time Analysis Tools: These windows allow you to view program activity in real-time. For example, the Execution Graph shows a diagram of thread activity.
- DSP/BIOS API: Your C and assembly language programs can call over 150 DSP/BIOS API functions.

This section contains the following lessons on getting started with DSP/BIOS:

- [Creating a DSP/BIOS Program](#)
- [Debugging Program Behavior](#)
- [Analyzing Real-Time Behavior](#)
- [Connecting to Virtual I/O Devices](#)



---

## Overview

[file:///C:/Documents and Settings/Pierre-Armand/Local Settings/Temp/~hhBA53.htm](#)

26/09/2007

Getting Started With the Code Composer Studio Tutorial

Page 318 sur 548

This lesson shows how to create a program that uses DSP/BIOS. The steps include creating a configuration file, adding files to a project, and adding DSP/BIOS API calls to a program.

Learning Objectives:

- Modify the project for DSP/BIOS
- Create configuration files
- Profile the project to measure performance improvement

### Examples used in this lesson: [hello1](#), [hello2](#)

Application Objective:

The example used in this lesson is a simple "hello world" program. Initially, it uses the standard C puts function. You modify the project to use DSP/BIOS functionality in this lesson. Using the profiling feature of Code Composer Studio, you measure the performance of both the standard puts function and the equivalent DSP/BIOS function.

This lesson consists of the following steps. To go directly to a step, click on the link below:

- [Opening and Examining the Project](#)
- [Profiling studio.h Execution Time](#)
- [Switching to the hello2 Source Code](#)
- [Creating a Configuration File](#)
- [Adding DSP/BIOS Files to a Project](#)
- [Testing with Code Composer Studio](#)
- [Profiling DSP/BIOS Code Execution](#)
- [Things to Try](#)



The forward arrow will take you to the next page in this lesson.

---

## Opening and Examining the Project

[file:///C:/Documents and Settings/Pierre-Armand/Local Settings/Temp/~hhBA53.htm](#)

26/09/2007

You begin this lesson by opening a project with Code Composer Studio and examining the source code files and libraries used in that project. Over the course of these lessons, you will learn about the following types of files:

- .lib This library provides runtime support for the target DSP chip
  - .c This file contains source code that provides the main functionality of this project
  - .h This file declares the buffer-C-structure as well as define any required constants
  - .jdt This file contains all of your project build and configuration options
  - .asm This file contains assembly instructions
  - .cmd This file maps sections to memory
1. Create a folder called `hellobios` in the `C:\CCStudio_v3.10\myprojects` folder.
  2. Copy all the files and folders from the `C:\CCStudio_v3.10\tutorial\target\hello1` folder to this new folder.
  3. If you have not already done so, from the Windows Start menu, choose Programs→Texas Instruments→Code Composer Studio 3.1.→Code Composer Studio 3.1. (Or, double-click the Code Composer Studio icon on your desktop.)
  4. Choose the Project→Open menu item. Open the `hello.jdt` project in the `C:\CCStudio_v3.10\myprojects\hellobios` folder. (If another project was already open in Code Composer Studio, right-click on that project's .jdt file in the Project View and choose Close from the context menu.)
  5. Code Composer Studio displays a dialog box indicating the library file was not found. This is because the project was moved. To locate this file, click the Browse button, and navigate to the compiler library folder (`C:\CCStudio_v3.10\c6000\cgtools\lib`). Select Object and Library Files (\*.\*, \*.\*) in the Files of type box. Select the `rt5.lib` file for the target you are configured for (`rt56200.lib` for this example) and click Open.
  6. Expand the Project View list by clicking the + signs next to Projects, `hello.jdt`, Libraries, and Source.
  7. Double-click on the `hello.c` program to open it. If the assembly instructions are shown, choose View→Mixed Source/ASM to hide the assembly code. Examine the [source code](#) for this program.



### Profiling studio.h Execution Time

---



file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007

Getting Started With the Code Composer Studio Tutorial

Page 320 sur 548

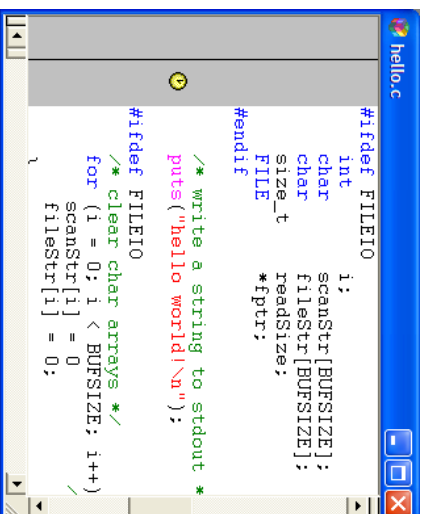
In this step, you use the profiling features of Code Composer Studio to measure the number of instruction cycles used by the calls to the puts function.

1. Choose Project→Rebuild All or click the  (Rebuild All) toolbar button.
2. Choose File→Load Program. In the Debug subfolder, select the program you just built, `hello.out`, and click Open. Because the call to puts writes to stdout, the Stdout tab area appears in the Code Composer Studio window.
3. Choose Profile→Setup. In the Profile Setup window, click on the Activities tab. If you see an error message that says a resource conflict occurred, [disable use of RTDX](#) .
4. Check the icons at the top of the Profile Setup tool to determine if profiling is enabled. The Enable/Disable Profiling icon  must be toggled on to enable profiling. Profiling may not be enabled if the program has started running or if it was explicitly disabled. Data will not be collected if profiling is not enabled.
5. Click on the box beside Collect Application Level Profile for Total Cycles and Code Size to collect this information. This activity measures the total cycles consumed by the entire application and calculates the total code size of the application.
6. Move to the Ranges tab of the Profile Setup window.
7. Double-click on the `hello.c` file in the main Project View window.
8. Highlight the call to the puts function, line 47. Your line number may vary.
9. Use your mouse to drag the highlighted text to the Ranges item in the Ranges tab of the Profile Setup window. A clock icon appears to the left of the line in `hello.c` to show the range to be profiled, which is a single line in this case.

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007





```

hello.c
#include <stdio.h>

int
main(
    char *argv,
    int argc)
{
    FILE *fp;
    char buf[BUFSIZ];
    int i;

    /* write a string to stdout */
    fputs("hello world\n");

    #ifndef FILEIO
    /* clear char arrays */
    for (i = 0; i < BUFSIZ; i++)
        scanStr[i] = 0;
    scanStr[0] = 0;
    #endif
}

```

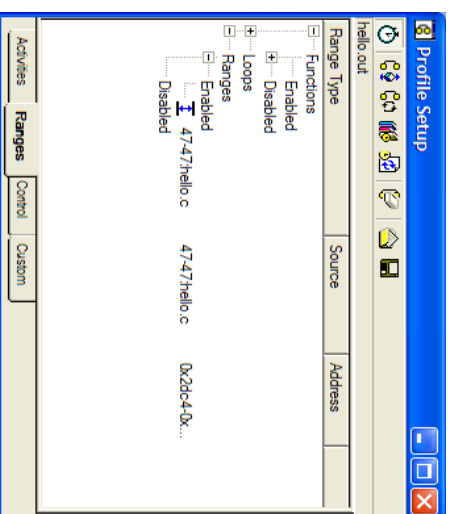
10. The Profile Setup window should look similar to the following. Your line number may vary.

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm


26/09/2007

Getting Started With the Code Composer Studio Tutorial

Page 322 sur 548

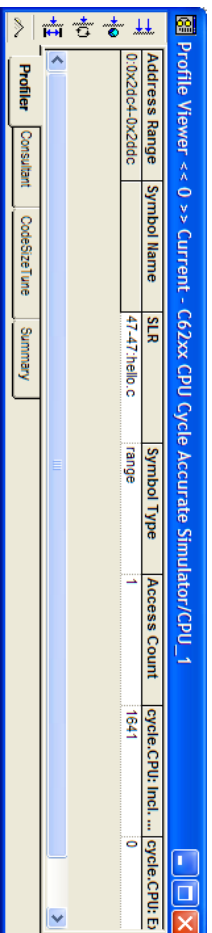


If you ever want to delete a line or range from the Profile window, select that item and press the Delete key.

11. From the Profile menu, choose Viewer to open the Profile Viewer on the bottom of the Code Composer Studio screen. It will be empty at this point.
12. Choose Debug → Run, or click the  (Run) toolbar button, or press F5 to run the program.
13. In the Profile Viewer window, click on the Profiler tab, and look at the cycle CPU:Incl>Total column. The Incl>Total column displays the number of cycles that occurred in the entire profiled section of code, including subroutines. This column is included in the data set because the cycle → Total event is selected when the Collect Application Level Profile for Total Cycles and Code Size activity is selected. Write the Incl>Total number down so that you can compare it to the number for the DSP/BIOS LOG\_printf function later in this lesson. The actual number of instruction cycles varies depending on your DSP platform.

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007



14. Before continuing, perform the following steps to **free the profiling resources** if you are using hardware to perform this tutorial. Disabling the Profiler is necessary because you use the RTDX to transfer DSP/BIOS information in the following steps. On most DSP platforms you cannot use both profiling and RTDX at the same time.
  15. Choose Debug →Halt to quit running the program.
  16. In the Profile Setup window, toggle off the Enable/Disable Profiling icon  to disable profiling.
  17. In the Profile→Clock menu, uncheck Enable Clock, if it is enabled.
  18. Close the Profile Viewer by right-clicking and choosing Close from the context menu.



### Switching to the hello2 Source Code

DSP / BIOS - 6S1

In this step, you copy a source file that uses a DSP/BIOS function to write the hello world message. The original hello.c program uses a standard C function, puts, to print the message to stdout. For programs that use DSP/BIOS, the LOG module provided with the DSP/BIOS API is a more efficient way to cause the target DSP to print messages on the host PC.

The DSP/BIOS API modules are optimized for use on real-time DSPs. Unlike C library calls such as puts, DSP/BIOS enables real-time analysis without halting your target hardware. Additionally, the API code consumes less space and runs faster than C standard I/O.

1. In Code Composer Studio, close the hello.c window.
2. Use the Windows Explorer to copy the hello.c file from the C:\CCStudio\_v3.10\tutorial\target\hello2 folder to your C:\CCStudio\_v3.10\myprojects\hellobios folder. Click **Yes** to replace the existing file. The hello2 example uses DSP/BIOS instead of the standard C puts function used by the hello1 example.

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007

### Getting Started With the Code Composer Studio Tutorial

Page 324 sur 548

3. Double-click on the hello.c program in the Project View.
4. Notice the following parts of the new **source code** :
  - The hello.c file first includes the std.h and log.h header files. Programs that use the DSP/BIOS API must include the std.h file and header files for any modules the program uses. The log.h header file defines the LOG\_OBJS structure and declares the API operations in the LOG module. The std.h file must be included before any DSP/BIOS module-specific header files. The order of the remaining DSP/BIOS header files is not important.
  - The code then includes the helloCfg.h header file, which will be generated when you create and save the configuration file in the **next step** of this lesson. This file contains external declarations for DSP/BIOS objects defined in the configuration file.
  - The helloCfg.h file also includes the DSP/BIOS module header files used by objects defined in the configuration file. Since the std.h and log.h files will be included by the helloCfg.h file, the lines that include them in the hello.c file are actually redundant. However, no problems are caused by including these header files twice.
  - The code calls LOG\_print and passes it the address of the LOG object (&trace) and the hello world message.
  - Finally main returns, which causes the program to enter the DSP/BIOS idle loop. Within this loop, DSP/BIOS waits for threads such as software interrupts and hardware interrupts to occur. In this example, there are no threads other than the idle loop.

**Note:** It is best to avoid using both LOG module functions and stdio functions in the same program. The stdio functions cause Code Composer Studio to use many internal breakpoints to move data between the target and host. In contrast, DSP/BIOS uses RTDX to move the data for real-time analysis. The breakpoints can interfere with these real-time analysis transfers on some platforms and in certain situations.



### Creating a Configuration File

DSP / BIOS - 6S1

In order to use the DSP/BIOS API, a program must have a DSP/BIOS configuration file. The configuration file defines objects and object properties for use by the application program.

In this step, you create a configuration file for the program. The configuration defines a LOG object used to display the "hello world" message.

1. Choose File→New→DSP/BIOS Configuration.
2. Select the **template** for your DSP target and click OK.

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007

3. You see the configuration window. The left side of the window lists categories of modules. You can see a list of **modules** by clicking the + symbol next to the category. You can see a module's objects by clicking the + symbol to the left of a module manager. The right side of the window shows properties of the selected manager or object. Click the + sign next to the Instrumentation category to display its list of modules.
4. Right-click on the LOG - Event Log Manager and choose Insert LOG from the context menu. This creates a LOG object called LOG0.
5. Right-click on the name of the LOG0 object and choose Rename from the context menu. Change the object's name to *trace*.



6. If you are using a simulator target and did not select a configuration template specifically for the simulator, **set the RTDX Mode to Simulator**.
7. Choose File-->Save. Move to your working folder (usually C:\CCStudio\_v3.10\myprojects\hellobios) and save this configuration file with a filename of hello.cdb. Saving this configuration generates the following files:
  - hello.cdb. Stores configuration settings
  - hello.cmd. Linker command file
  - hellocfg.h. Includes DSP/BIOS module header files and declares external variables for objects created in the configuration file
  - hellocfg.s62. Assembly language source file for DSP/BIOS settings
  - hellocfg.h62.. Assembly language header file included by hellocfg.s62.
  - hellocfg.c.c. Code for Chip Support Library (CSL) structures and settings

You may notice a hello.cdf file in your project directory. This is a temporary file that exists only when the configuration file is open. Do not attempt to open, delete, or modify this file.



### Adding DSP/BIOS Files to a Project

#### DSP/BIOS - GSI

Recall that saving the configuration you created in the previous step actually created several new files, including hello.cdb and hellocfg.cmd. In this section, you add these files to the project.


file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007

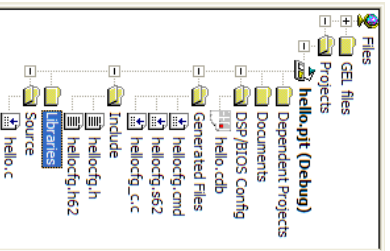
### Getting Started With the Code Composer Studio Tutorial

Page 326 sur 548

and remove the files they replace.

1. Choose Project-->Add Files to Project. Select Configuration File (\*.cdb) in the Files of type box. Select the hello.cdb file and click Open. Notice that the Project View now contains hello.cdb in a folder called DSP/BIOS Config. In addition, Code Composer Studio automatically adds the hellocfg.s62., hellocfg.cmd., and hellocfg.c.c files to the Generated Files folder.
2. The output filename must match the .cdb filename (hello.out and hello.cdb). Choose Project-->Build Options and go to the Linker tab. Verify that the Output Filename field says .Debug\hello.out. Click OK.
3. Click on the hello.cmd linker command file in Project View, and choose Remove from Project in the context menu. If you want your own projects to include a separate linker command file, you can create your own linker command file. This file must begin by including the linker command file created by the Configuration Tool.
4. In the Project View area, right-click on the vectors.asm source file and choose Remove from Project in the context menu. The hardware interrupt vectors are automatically defined by the DSP/BIOS configuration file.
5. Right-click on the ts6200.lib library file and choose Remove from Project in the context menu. This library is automatically included by the hellocfg.cmd file.
6. Choose Project-->Save to save your changes to the project. It is a good idea to save changes to a project before building or running a program.
7. Choose Project-->Rebuild All or click the  (Rebuild All) toolbar button.

The changes you made should affect the project view, as shown here:



file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007



### Testing With Code Composer Studio

---

**DSP/BIOS - GS1**

Now you can test the program. Since the program writes one line to a DSP/BIOS log, testing is simple.

1. Choose File→Load Program. In the Debug subfolder, select the program you just built, hello\_out, and click Open.
2. Choose DSP/BIOS→Message Log. A Message Log area appears at the bottom of the Code Composer Studio window.
3. If it is not already selected, select trace in the Log Name list.
4. Right-click on the Message Log window and select Property Page from the context menu.
5. In the Message Log Properties window, put a checkmark in the Log to file box and type hello.txt as the filename. Then click OK.
6. Choose Debug→Run or press F5. The hello world message appears in the Message Log area. The message may take a second to appear in the Message Log area because data stored in the log on the target is sent to the host PC only once per second. (To change the refresh rate, choose DSP/BIOS→RTA Control Panel; Right-click on the RTA Control Panel area and choose Property Page. Change the message Log refresh rate and click OK.)
7. Choose Debug→Halt or press Shift F5 to stop the program. After the main function returns, your program is in the DSP/BIOS idle loop, waiting for an event to occur. See [Things to Try](#) to learn more about the idle loop.
8. Close the Message Log by right-clicking and selecting Close.
9. Choose File→Open and select Text Files (\*.txt) in the Files of type list. Find and open the hello.txt file. The file will be in the C:\CCStudio\_V3.10\myprojects\hellobios\Debug folder, unless your working folder is located elsewhere or you opened a file from another folder after loading the program.
10. Notice that the hello.txt file contains the same hello world message.
11. Choose Tools→RTDX→Configuration Control to open the RTDX Configuration Control window. Remove the checkmark from the Enable RTDX checkbox. Then right-click on the RTDX area and select Close. Disabling RTDX is necessary because you use the Profiler in the next section. On most DSP platforms you cannot use both [profiling](#) and RTDX at the same time.



file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007

### Getting Started With the Code Composer Studio Tutorial


Page 328 sur 548

### Profiling DSP/BIOS Code Execution

---

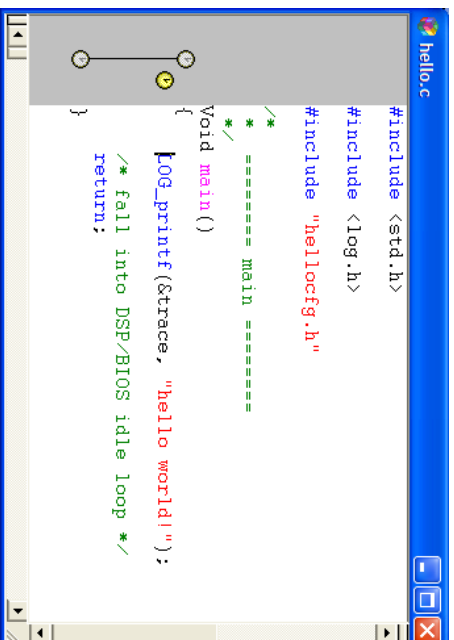
**DSP/BIOS - GS1**

In this step, you use the profiling features of Code Composer Studio to measure the number of instruction cycles used by the calls to LOG\_print.

1. Choose File→Reload Program.
2. Choose Profile→Setup. In the Profile Setup window, click on the Activities tab. If you see an error message that says a resource conflict occurred, [disable use of RTDX](#) .
3. Check the icons at the top of the Profile Setup tool to determine if profiling is enabled. The Enable/Disable Profiling icon  must be toggled on to enable profiling. Profiling may not be enabled if the program has started running or if it was explicitly disabled. Data will not be collected if profiling is not enabled.
4. Click on the box beside Collect Application Level Profile for Total Cycles and Code Size to collect this information. This activity measures the total cycles consumed by the entire application and calculates the total code size of the application.
5. Move to the Ranges tab of the Profile Setup window.
6. Double-click on the hello.c file in the main Project View window.
7. Highlight the call to the LOG\_print function.
8. Use your mouse to drag the highlighted text to the Ranges item in the Ranges tab of the Profile Setup window. A clock icon appears to the left of the line to show the range to be profiled, which is a single line in this case.

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007



```

hello.c
#include <std.h>
#include <log.h>
#include "hellocfg.h"

/*
 * ===== main =====
 */
void main()
{
    LOG_printf(&trace, "hello world!");
    /* fall into DSP/BIOS idle loop */
    return;
}

```

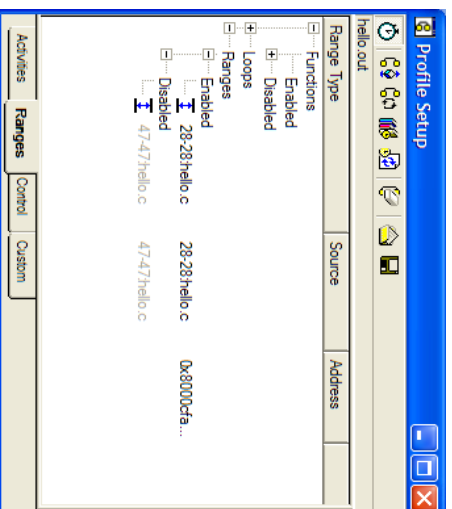
- The Profile Setup window should look similar to the following. The actual line number may be different. (If you ever want to delete a line or range from the Profile window, select the row for that item and press the Delete key.)



file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007

Getting Started With the Code Composer Studio Tutorial

Page 330 sur 548



- If the Profile Viewer is not still open, From the Profile menu, choose Viewer to open the Profile Viewer on the bottom of the CCSstudio screen. It will be empty at this point.
- Choose Debug→Run, or click the  (Run) toolbar button, or press F5 to run the program.
- In the Profile Viewer window, look at the number in the cycle.CPU:ind Total column. Compare the number of instructions for the LOG\_printf function to the number you measured earlier for the puts function. Notice that LOG\_printf is much more efficient than the puts function. The actual number of instruction cycles varies depending on your DSP platform. You may need to choose halt and Reset CPU from the Debug menu to view the new profiling information in the Profile Viewer window.
- Calls to LOG\_printf are efficient because string formatting is performed on the host PC rather than on the target DSP. Because LOG\_printf is so efficient, you can leave calls to LOG\_printf in your code for system status monitoring with very little impact on code execution.
- Before continuing to the next lesson (after completing [Things to Try](#)), perform the following steps to [free the profiling resources](#) and prepare for the next lesson.
  - In the Profile Setup window, toggle off the Enable/Disable Profiling icon  to disable profiling.
  - In the Profile→Clock menu, uncheck Enable Clock, if the clock was enabled.
  - Close the Profile Viewer by right-clicking and choosing Close from the context menu.

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007

17. Close the project and all Code Composer Studio windows you were using for testing.



## Things to Try

---

### DSP/BIOS - G51

To explore Code Composer Studio, try the following:

- Load `hello.out` and put a breakpoint on the line that calls `LOG_print`. Use `Debug->Breakpoints` to add a breakpoint at `IDL_F_loop`. (Type `IDL_F_loop` in the location box and click `Add`.) Run the program. At the first breakpoint, use `View->CPU Registers->Core Registers` to see a list of register values. Notice that `GIE` is 0, indicating that interrupts are disabled while the main function is executing. Run to the next breakpoint. Notice that `GIE` is now 1, indicating that interrupts are now enabled. Notice that if you run the program, you hit this breakpoint over and over.

After the startup process and main are completed, a DSP/BIOS application drops into a background thread called the idle loop. This loop is managed by the `IDL` module and continues until you halt the program. The idle loop runs with interrupts enabled and can be preempted at any point by any hardware interrupt, software interrupt, or task triggered to handle the application's real-time processing.

- In an MS-DOS window, run the `secttl.exe` utility on `hello.out` and direct the output to `hello1.prn`. Open `hello1.prn` in a text editor. `Secttl` lists the code and data sections in the `*.out` files.

Compare the `hello1.prn` and `hello2.prn` files to see differences in memory sections and sizes when using `stdio.h` calls and DSP/BIOS. Notice the size of the text section is smaller for the DSP/BIOS call to `LOG_print`, as compared to linking the `stdio` when using `puts()`. See the `secttl` utility topic for more information on the `secttl` utility.

This concludes Creating a DSP/BIOS Program. The next lesson will cover debugging program behavior.



## Overview

---

### DSP/BIOS - G52

This lesson shows how to modify an example program to cause that program to schedule its functions and allow multi-threading. You view performance information for debugging purposes. You also use more features of DSP/BIOS, including the Execution Graph, the real-time analysis control panel (RTA Control Panel), the Statistics View, and the CLK (clock), SWI (software

<file:///C:/Documents and Settings/Pierre-Armand/Local Settings/Temp/~hhBA53.htm>

26/09/2007

Getting Started With the Code Composer Studio Tutorial

Page 332 sur 548

Interrupt), STS (statistics), and TRC (trace) modules.

Learning Objectives:

- Modify the project for DSP/BIOS
- Modify the configuration file
- Use DSP/BIOS tools to debug the project

### Examples used in this lesson: volume2

**Note:** If you are using a G6416 target, DSP/BIOS real-time analysis data is available in stop-mode only. That is, analysis data is transferred from the target to the host PC only when you reach a breakpoint or halt the program. If you are using an XDS560 to connect to your DSK, DSP/BIOS real-time analysis data is available in stop-mode only. That is, analysis data is transferred from the target to the host PC only when you reach a breakpoint or halt the program.

Application Objective:

The example used in this lesson performs some simple signal processing. It multiplies each amplitude value in the input buffer by the gain and puts the resulting values into the output buffer. It also uses a load routine written in assembly to consume instruction cycles based on the processingload value passed to the routine.

This lesson consists of the following steps. To go directly to a step, click on the link below:

[Opening and Examining the Project](#)

[Reviewing the Source](#)

[Modifying the Configuration File](#)

[Viewing Thread Execution](#)

[Changing and Viewing the Load](#)

[Analyzing Thread Statistics](#)

[Adding Explicit STS Instrumentation](#)

[Viewing Explicit Instrumentation](#)

[Things to Try](#)



<file:///C:/Documents and Settings/Pierre-Armand/Local Settings/Temp/~hhBA53.htm>

26/09/2007

The forward arrow will take you to the next page in this lesson.

## Opening and Examining the Project

### DSP/BIOS - G52

You begin this lesson by opening a project with Code Composer Studio and examining the source code files and libraries used in that project.

1. Create a folder called `volume2` in the `C:\CCStudio_v3.10\myprojects` folder.
2. Copy all the files and folders from the `C:\CCStudio_v3.10\tutorial\target` `volume2` folder to this new folder.
3. If you have not already done so, from the Windows Start menu, choose Programs→Texas Instruments→Code Composer Studio 3.1→Code Composer Studio 3.1. (Or, double-click the Code Composer Studio icon on your desktop.) If another project was already open in Code Composer Studio, right-click on that project's .pj1 file in the Project View and choose Close from the shortcut menu.
4. Choose Project→Open. Select the `volume.pj1` file in the folder you created and click Open.
5. Expand the Project View by clicking the + signs next to Projects, `volume.pj1`, Generated Files, DSP/BIOS Config, and Source. The `volumecfg.cmd` file, which was created along with a configuration file, includes a large number of DSP/BIOS header files. (You do not need to examine all these header files.)

Over the course of the lesson, you will use the following files:

- `volume.cdb`: This is the configuration file for the project.
- `volume.c`: This is the source code for the main program. You examine the source code in the next section.
- `volume.h`: This is a header file included by `volume.c` to define various constants and structures. It is identical to the `volume.h` file used in Lesson 2 of the Code Composer Studio module.
- `load.asm`: This file contains the load routine, a simple assembly loop routine that is callable from C with one argument. It is identical to the `load.asm` file used in Lesson 2 of the Code Composer Studio module.
- `volumecfg.cmd`: This linker command file is created when you save the configuration file.
- `volumecfg.h`: This header file is created when you save the configuration file. It includes DSP/BIOS module header files and declares external variables for objects created in the configuration file.
- `volumecfg.s62`: This assembly file is created when you save the configuration file and contains DSP/BIOS configuration settings.
- `volumecfg.h62`: This header file is created when you save the configuration file and is included by `volumecfg.s62`.

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007

Getting Started With the Code Composer Studio Tutorial

Page 334 sur 548

- `volumecfg.c`: This file is created when you save the configuration file. It contains code for Chip Support Library (CSL) structures and settings.

## Reviewing the Source Code

### DSP/BIOS - G52

This example shows how an application using DSP/BIOS can schedule real-time behavior and manage data transfer triggered by external events. Rather than having the main function loop forever, the data I/O in the real application is likely to happen as a result of a periodic external interrupt. A simple way to simulate a periodic external interrupt in the example is to use the timer interrupt from the on-chip timer.

Double-click on the `volume.c` file in the Project View to see the [source code](#) in the right half of the Code Composer Studio window. Notice the following aspects of the example:

The data types for declarations are capitalized. DSP/BIOS provides data types that are portable to other processors. Most types used by DSP/BIOS are capitalized versions of the corresponding C types.

- The code uses #include to reference three DSP/BIOS header files: `std.h`, `log.h`, and `swi.h`. The `std.h` file must be included before other DSP/BIOS header files.
  - The code then includes the `volumecfg.h` header file, which will be generated when you modify and save the configuration file in the [next step](#) of this lesson. This file contains external declarations for DSP/BIOS objects defined in the configuration file.
  - The main function simply returns after calling `LOG_print` to display a message. When the main function returns, the program drops into the DSP/BIOS idle loop. At this point, the DSP/BIOS scheduler manages thread execution.
  - The processing function is called by a software interrupt called `processing_SWI`, which yields to all hardware interrupts. DSP/BIOS provides a number of [thread types](#), including hardware interrupts, software interrupts, tasks, and idle threads.
- Instead of using a software interrupt (SWI), a hardware interrupt could perform the signal processing directly. However, signal processing may require a large number of cycles, possibly more than the time until the next interrupt. Such processing would prevent the interrupt from being handled.
- The `dataIO` function calls `SWI_dec`, which decrements a counter associated with a software interrupt object. When the counter reaches 0, the software interrupt schedules its function for execution and resets the counter.

The `dataIO` function simulates hardware-based data I/O. A typical program accumulates data in a buffer until it has enough data to process. In this example, the `dataIO` function is performed 10 times for each time the processing function is performed. The counter decremented by `SWI_dec` controls this.



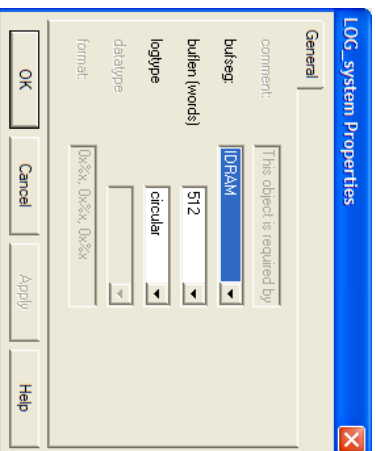
## Modifying the Configuration File

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007

A DSP/BIOS configuration file has already been created for this example. In this section, you examine objects that have been added to the default configuration.

1. In the project View, double-click on the volume.cdb file (in the DSP/BIOS Config folder) to open it.
2. Click the + sign next to the Instrumentation and Scheduling categories to display their lists of modules.
3. Click the + signs next to the LOG - Event Log Manager, CLK - Clock Manager, and SWI - Software Interrupt Manager. This configuration file contains objects for these modules in addition to the default set of objects in the configuration file.
4. Highlight the LOG object called trace. You see the properties for this log in the right half of the window. This object's properties are the same as those of the trace LOG you created for the configuration file in [Creating a DSP/BIOS Program](#). The volume.c program calls LOG\_print to write volume example started to this log.
5. Right-click on the LOG object called LOG\_system. From the context menu, select Properties. You see the Properties dialog for this object. When you run the program, this log stores events traced by the system for various DSP/BIOS modules. This is a generic dialog, your dialog properties may vary depending on your device.



6. Change the buflen property to 256 words and click OK.
7. Highlight the CLK object called dataIO\_CLK. Notice that the function called when this CLK object is activated is \_dataIO. This is the dataIO function in volume.c.

**Note:** Underscores and C Function Names: This C function name is prefixed by an underscore because saving the configuration generates assembly language files. The underscore prefix is the convention for accessing C functions from assembly. (See the section on Interfacing C with assembly language in the "TMS320Cx Optimizing C Compiler User's Guide" for more information.)

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

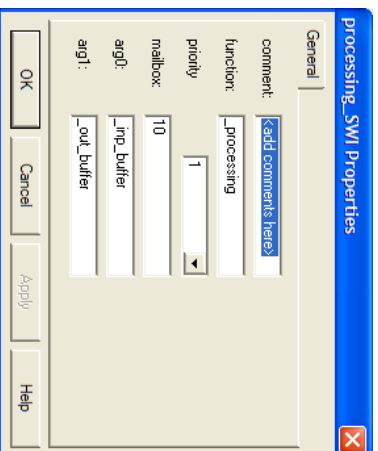
26/09/2007

Getting Started With the Code Composer Studio Tutorial

Page 336 sur 548

This rule applies only to C functions you write. You do not need to use the underscore prefix with configuration-generated objects or DSP/BIOS API calls because two names are automatically created for each object: one prefixed with an underscore and one without.

8. Since the dataIO function is no longer run within main, what causes this CLK object to run its function? To find out, right-click on the CLK - Clock Manager. From the context menu, select Properties. You see the Clock Manager Properties dialog. Notice that the Microseconds/Int property is 1000. This means the clock functions, including dataIO\_CLK, run once a millisecond. Notice that the CPU Interrupt property is shown in gray. This is because the property is actually set in the Properties dialog of the corresponding HWT object.
9. Click Cancel to close the Clock Manager Properties dialog without making any changes.
10. Expand the list of HWI objects and right-click on the hardware object that was listed in the CPU Interrupt field for the CLK Manager Properties (HWI\_INT14, in this case). From the context menu, select Properties. This object runs a function called CLK\_F\_isr when the on-chip timer causes an interrupt. The CLK object functions run from the context of this CLK\_F\_isr hardware interrupt function. Therefore, they run to completion without yielding and have higher priority than any software interrupts. (The CLK\_F\_isr saves the register context, so the CLK functions do not need to save and restore context as would normally be required within a hardware interrupt function.)
11. Click Cancel to close the Properties dialog without making any changes.
12. Expand the list of SWI objects and open the Properties dialog for the processing\_SWI software interrupt object.



- function. When this software interrupt is activated, the processing function runs. This function is shown in [Reviewing the Source Code](#).
- mailbox. The mailbox value can control when a software interrupt runs. Several API calls affect the value of the mailbox and can post the software interrupt depending on the resulting value. When a software interrupt is posted, it runs when it is the highest priority software or hardware interrupt thread that has been posted.
- arg0, arg1. The inp\_buffer and out\_buffer addresses are passed to the processing function.

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007



- Click Cancel to close the Properties dialog without making any changes.

Since the processing function is no longer run within main, what causes this SWI object to run its function? In volume.c, the dataIO function calls SWI\_dec, which decrements the mailbox value and posts the software interrupt if the new mailbox value is 0. So, this SWI object runs its function every tenth time the dataIO\_CLK object runs the dataIO function, or once every 10 milliseconds.

- Choose File→Close. You will be prompted to save your changes to volume.cdb. Click Yes. Saving this file also generates the following files: volumecfg.cmd, volumecfg.h, volumecfg.h62., volumecfg.s62., and volumecfg.c.c.
- Choose Project→Build or click the  (Incremental Build) toolbar button.



### Viewing Thread Execution with the Execution Graph

DSP/BIOS - G52

You could test this program by putting a Probe Point within the processing function and viewing graphs of input and output data. (To see how, go to the [Data Visualization](#) lesson in the Code Composer Studio IDE module of the tutorial.)

However, in order to focus on the real-time aspects of DSP/BIOS, let us assume you have already tested the signal processing algorithm. At this stage of development, your focus should be on making sure threads can meet their real-time deadlines.

- Choose File→Load Program. In the Debug subfolder of the volume2 folder, select the program you just built, volume.out, and click Open.
- Choose Debug→Go Main. The program runs to the first statement in the main function.
- Choose DSP/BIOS→RTA Control Panel. You see a window with checkboxes for the instrumentation types provided by DSP/BIOS. By default, all of the instrumentation types are enabled.
- Verify that the Enable SWI logging and Enable CLK logging boxes are checked. Scroll down and verify that the Global host enable box is checked.
- Choose DSP/BIOS→Execution Graph. [Resize](#) the Execution Graph window as needed.
- Right-click on the RTA Control Panel and choose Property Page from the context menu.
- Verify that the Refresh Rate for Message Log/Execution Graph is 1 second and click OK.

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

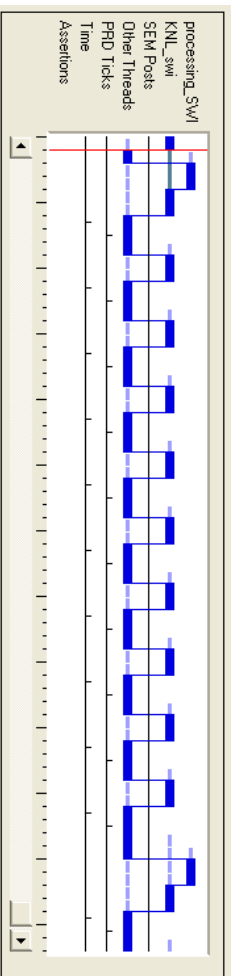
26/09/2007

### Getting Started With the Code Composer Studio Tutorial

Page 338 sur 548

- Choose Debug→Run or click the  (Run) toolbar button.

The Execution Graph should look similar to this, although it may vary depending on your device:



- The marks in the Time row show each time the Clock Manager ran the CLK functions. Count the marks between times the processing\_SWI object was running. There should be 10 marks. This indicates that the processing\_SWI object ran its function every tenth time the dataIO\_CLK object ran its function. This is as expected because the mailbox value that is decremented by the dataIO function starts at 10.



### Changing and Viewing the Load

DSP/BIOS - G52

Using the Execution Graph, you saw that the program meets its real-time deadlines. However, the signal processing functions in a typical program must perform more complex and cycle-consuming work than multiplying a value and copying it to another buffer. You can simulate such complex threads by increasing the cycles consumed by the load function.

Note: The CPU Load Graph does not function with simulators, this example will only fully function using hardware.

Note: The following load values are for a C running at 100 MIPS. If your DSP is running at a different speed, you need to multiply the load values by (Your MIPS / 100). If you do not know what MIPS you are running at, open volume.cdb and look at the Global Settings property called DSP Speed in MHz (CLKOUT). The Global Settings manager is in the System Settings category.

- Choose DSP/BIOS→CPU Load Graph. A blank CPU Load Graph appears. This graph will not function with simulators.
- Right-click on the RTA Control Panel and choose Property Page from the context menu.
- Change the Refresh Rate for Statistics View/CPU Load Graph to 0.5 seconds and click OK.

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

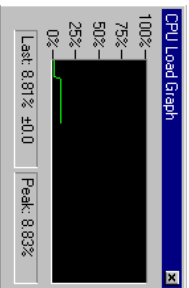
26/09/2007

Notice that the CPU load is currently very low.

If the CPU load is not shown, choose Debug→Halt to pause the target and update the analysis data shown in the graph. If the CPU load is still not shown, you may be using a simulator that does not support CPU load calculations.

The Statistics View and the CPU Load transfer very little data from the target to the host, so you can set these windows to update frequently without causing a large effect on program execution. The Message Log and Execution Graph transfer the number of words specified for the buflen property of the corresponding LOG object in the configuration file. Since more data is transferred, you may want to make these windows update less frequently.

4. Choose File→Load GEL. In the Load GEL File dialog, select the volume.gel file in the volume2 folder and click Open.
5. Choose GEL→Application Control→Load.
6. Type 100 as the new load and click Execute. The CPU load increases to about 9%.



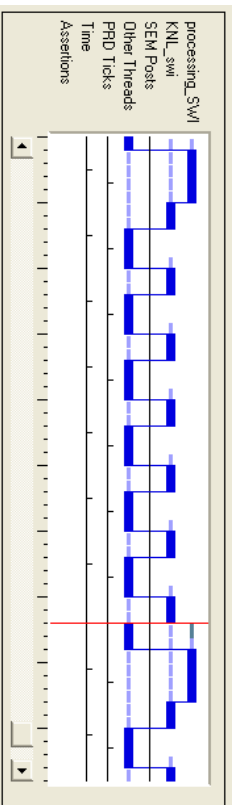
7. Right-click on the Execution Graph and choose Clear from the context menu. Notice that the program still meets its real-time deadline. There are 10 time marks between each execution of the processing\_SWI function.
8. Using the GEL control, change the load to 200 and click Execute.
9. Right-click on the Execution Graph and choose Clear from the context menu. One or two of the time marks occur while the processing\_SWI function is executing. Does this mean the program is missing its real-time deadline? No, it shows that the program is functioning correctly. The hardware interrupt that runs the CLK object functions can interrupt the software interrupt processing, and the software interrupt still completes its work before it needs to run again. Your graph may vary depending on your device.

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

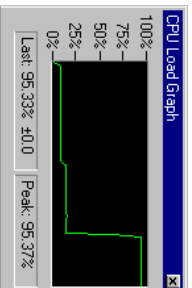
26/09/2007

Getting Started With the Code Composer Studio Tutorial

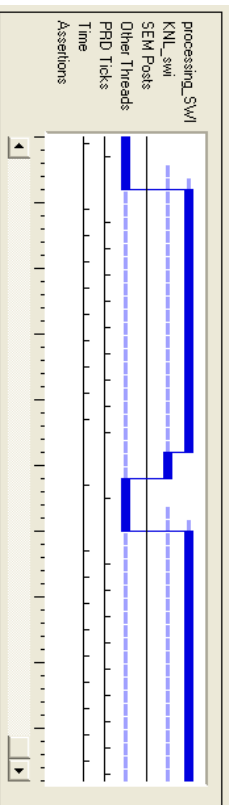
Page 340 sur 548



10. Using the GEL control, change the load to 1250 and click Execute. The CPU load increases to about 95% and the Execution Graph and CPU load are updated less frequently.



11. Right-click on the Execution Graph and choose Clear from the context menu. The program still meets its real-time deadline because processing\_SWI completes before 10 time marks have occurred. Your graph may vary depending on your device.

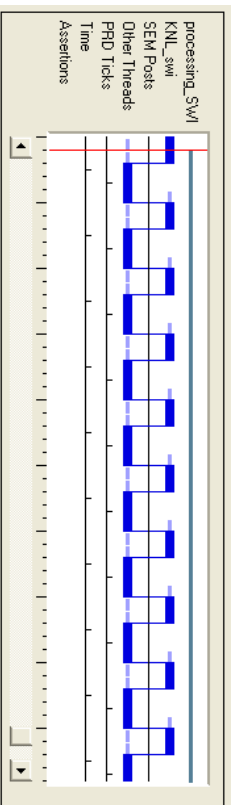
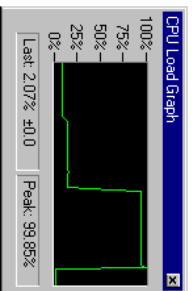


12. Using the GEL control, change the load to 1500 and click Execute. The CPU Load Graph and the Execution Graph stop updating. This is because Execution Graph data is sent to the host within an idle thread, which has the lowest execution priority within the program. Because the higher-priority threads are using all the processing time, there is not enough time for the host control updates to be performed. The program is now missing its real-time deadline.

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007

13. Choose Debug→Halt. This stops the program and updates the Execution Graph. You can see that processing\_SWI runs far too long. You may see squares in the Assertions row. These squares indicate that the Code Composer Studio detected that the application missed a real-time deadline.
14. Using the GEL control, change the load to 10 and click Execute.
15. Choose Debug→Run. The CPU load and Execution Graph begin updating again. Your graph may vary depending on your device.



Note : Using the Load GEL control temporarily halts the target. If you are analyzing a real-time system and do not want to affect system performance, modify the load using a Real-Time Data Exchange (RTDX) application. The next lesson shows how to modify the load in real-time using RTDX.



### Analyzing Thread Statistics

DSP/BIOS - G52


You can use other DSP/BIOS controls to examine the load on the DSP and the processing statistics for the processing\_SWI object.

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007

### Getting Started With the Code Composer Studio Tutorial

Page 342 sur 548

1. Choose DSP/BIOS→Statistics View. A Statistics View area appears. In the RTA Control Panel, verify that the Enable SWI accumulators box is checked.
  2. If you have halted the program, click the  (Debug→Run) toolbar button.
  3. You see statistics for various objects. Resize the window so that you can see all the columns.
  4. Notice the Max value for processing\_SWI in the Statistics View. SWI statistics are shown in units of instruction cycles by default. The actual statistics are different from the figure shown below on different DSP platforms. However, notice that the Max value does not change while the load value is constant. Your graph may vary depending on your device.
- | STIS           | Count | Total          | Max              | Average   |
|----------------|-------|----------------|------------------|-----------|
| processing_SWI | 402   | 263612408 inst | 1506052 inst     | 655752.26 |
| TSK_idle       | 0     | 0 inst         | -2147483648 inst | 0.00      |
| IDL_busyDbl    | 8919  | -572774        | -63              | -64.9477  |
5. Using the GEL control, increase the load and click Execute. Notice the change in the Max value for the number of instructions performed from the beginning to the end of processing\_SWI increases.
  6. Right-click on the Statistics View area and choose Property Page from the context menu. In the Units tab, select Milliseconds as the unit for the processing\_SWI object. In order to meet its real-time deadline, processing\_SWI must complete in less than 10 milliseconds. (If processing\_SWI is not shown in the Units tab, make sure processing\_SWI is selected in the General tab.)
  7. Experiment with different load values. If you decrease the load, right-click on the Statistics View and select Clear from the context menu. This resets the fields to their lowest possible values, allowing you to see the current value in the Max field.
  8. Choose Debug→Halt to stop running and close all the DSP/BIOS and GEL controls you have opened.



### Adding Explicit STS Instrumentation

DSP/BIOS - G52

In the previous section, you used the Statistics View to see the number of instructions performed during a software interrupt's execution. If you use a configuration file, DSP/BIOS supports

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007

such statistics automatically. This is called implicit instrumentation. You can also use API calls to gather other statistics. This is called explicit instrumentation.

1. In the Project View, double-click on the volume.cdb file (in the DSP/BIOS Config folder) to open it.
2. Click the + sign next to the Instrumentation category to display its list of modules.
3. Right-click on the STS - Statistics Object Manager and choose Insert STS from the context menu. The default name for the created STS object should be changed to `processingload_STIS`.

4. Choose File→Close. You are asked whether you want to save your changes to volume.cdb. Click Yes.

5. In the Project View, double-click on the volume.c file to open it for editing. Make the following changes to the file. (Click [here](#) to see the full program including the changes.)

```

■ Add the following lines below the line that includes the swi.h file:
#include <clk.h>
#include <sis.h>
#include <trc.h>

```

■ Add the following lines within the processing function before the call to the load function:

```

/* enable instrumentation only if TRC_USER0 is set */
if (TRC_query(TRC_USER0) == 0) {
    STS_set(&processingload_STIS, CLK_gettime());
}

```

■ Add the following lines within the processing function after the call to the load function:

```

if (TRC_query(TRC_USER0) == 0) {
    STS_delta(&processingload_STIS, CLK_gettime());
}

```

The calls to TRC\_query with the TRC\_USER0 constant cause the program to perform the STS\_set and STS\_delta calls only if you enable the USER0 trace in the RTA control panel (or programmatically by calling TRC\_enable).

6. Choose File→Save to save your changes to volume.c.
7. Choose Project→Build or click the  (Incremental Build) toolbar button.



file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007

Getting Started With the Code Composer Studio Tutorial

Page 344 sur 548

### Viewing Explicit Instrumentation

#### DSP/BIOS - G52

To view information provided by the explicit instrumentation calls you added, you use the Statistics View.

**Note:** The values in this section are for a C6000 running at 100 MIPS. If your DSP is running at a different speed, multiply processingload\_STIS values by (Your MIPS / 100). To find your MIPS value, look at the Global Settings property called DSP Speed in MHz (CLKOUT) in the configuration file. The Global Settings Manager is in the System category.

1. Choose File→Load Program. In the Debug subfolder, select the program you just rebuilt, volume.out, and click Open.
2. Choose DSP/BIOS→RTA Control Panel.
3. Verify that the Enable SWI accumulators, Enable USER0 trace, and Global host enable boxes are checked. Enabling USER0 tracing causes the calls to TRC\_query(TRC\_USER0) to return 0.
4. Choose DSP/BIOS→Statistics View.
5. Right-click on the Statistics View area and choose Property Page from the context menu. In the General tab, disable all objects except the processing\_SWI and processingload\_STIS objects. Only the processing\_SWI and processingload\_STIS should be enabled when you are finished with this step.
6. Go to the Units tab and select the processingload\_STIS object and type "inc" (for increment) as the unit label. Since the calls to STS\_set and STS\_delta use the CLK\_gettime function, the statistics are reported in timer increments.
7. In the Units tab, select the processing\_SWI object and select Instructions as the unit.
8. Click OK. You see the statistics fields for the two objects you selected. Resize the window so that you can see all the columns.

9. Choose Debug→Run or click the  (Run) toolbar button.

10. Multiply the Max value for processingload\_STIS by 4.

SWI statistics are measured in instruction cycles. Because you used the CLK\_gettime function to benchmark the processing load, processingload\_STIS statistics are measured in on-chip timer counter increments. This counter is incremented at the following rate, where CLKOUT is the DSP clock speed in MIPS (see the Global Settings Property dialog) and TDDR is the value of the timer divide-down register (see the CLK Manager Property dialog):  $\text{CLKOUT} / (\text{TDDR} + 1)$

On TMS320CG6000 DSPs, the high-resolution time is incremented every 4 instruction cycles. Therefore, to convert the processingload\_STIS units to instruction cycles, you multiply by 4.

11. Subtract the resulting processingload\_STIS Max value in instruction cycles from the processing\_SWI Max value. The result varies depending on your DSP platform and development target. These instructions are performed within the processing function, but not between the calls to STS\_set and STS\_delta, as shown below.

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007

```
processing_SW1
processingLoad_STS
}

/* ===== processing ===== */
int processing(int *input, int *output)
{
    int size = BUFSIZE;
    while(size--){
        *output++ = *input++ * gain;
    }
    /* enable instrumentation if TRC_USER0 is set */
    if (TRC_query(TRC_USER0) == 0) {
        STS_set(&processingLoad_STS, CLK_gettime());
    }
    /* additional processing load */
    load(&processingLoad);
    if (TRC_query(TRC_USER0) == 0) {
        STS_delta(&processingLoad_STS, CLK_gettime());
    }
    return (TRUE);
}
```

For example, suppose the load is 10, the processingLoad\_STS Max is 2604, and the processing\_SW1 Max is 13832. To calculate the instruction cycles performed within the processing function but outside the calls to STS\_set and STS\_delta, the equation would be:

$$13832 - (2604 * 4) = 3416$$

12. Choose GEL→Application Control→Load. (If you have closed and restarted Code Composer Studio, you must reload the GEL file.)
13. Change the Load and click Execute.
14. Notice that while both Max values increase, the difference between the two Max values (after you multiply the processingLoad\_STS Max by 4) stays the same.
15. Remove the check mark from the Enable USER0 trace box in the RTA Control Panel.
16. Right-click on the Statistics View and choose Clear from the context menu.
17. Notice that no values are updated for processingLoad\_STS. This is because disabling the USER0 trace causes the following statement in the program to be false:  
if (TRC\_query(TRC\_USER0) == 0)  
As a result, the calls to STS\_set and STS\_delta are not performed.
18. Before continuing to the next lesson (after completing [Things to Try](#)), perform the following steps to prepare for the next lesson:

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007

Getting Started With the Code Composer Studio Tutorial

Page 346 sur 548

19. Click Debug→Halt or press Shift F5 to stop the program.
20. Close all GEL dialog boxes, DSP/BIOS analysis tools, and source windows.



### Things to Try

**DSP/BIOS - G52**

To further explore DSP/BIOS, try the following:

- Change the Host Operation property of the processingLoad\_STS object in the configuration file to A \* x and the A property to 4. This Host Operation multiplies the statistics by 4. The calls to CLK\_gettime cause statistics to be measured in high-resolution timer increments, which occur every 4 CPU cycles. So, changing the Host Operation converts the timer increments to CPU cycles. Rebuild the program and notice how the values in the Statistics View change.
- Modify volume.c by using the CLK\_gettime function instead of the CLK\_gettime function. Rebuild the program and notice how the values in the Statistics View change. The CLK\_gettime function gets a low-resolution time that corresponds to the timer marks you saw in the Execution Graph. You must increase the load significantly to change the Statistics View values when using CLK\_gettime.

This concludes the Debugging Program Behavior lesson. The next lesson will cover analyzing real-time behavior.



### Overview

**DSP/BIOS - G53**

This lesson shows how to analyze real-time behavior and correct scheduling problems using the example from the previous lesson. You use RTDX (Real-Time Data Exchange) to make real-time changes to the target, use DSP/BIOS periodic functions, and set software interrupt priorities.

Learning Objectives:

- Modify the project for DSP/BIOS
- Use RTDX to make real-time changes to the target

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007

- Use DSP/BIOS tools to measure performance

**Examples used in this lesson: volume2, volume4****Application Objective:**

The example used in this lesson builds on the example used in the previous lesson. Instead of using a GEL control to set the load, it uses RTDX to change the load of the processing signal in real time.

This lesson consists of the following steps. To go directly to a step, click on the link below:

[Opening and Examining the Project](#)

[Modifying the Configuration File](#)

[Reviewing the Source Code Changes](#)

[Using the RTDX to Change the Load](#)

[Modifying Software Interrupt Priorities](#)

[Things to Try](#)



The forward arrow will take you to the next page in the lesson.

**Opening and Examining the Project****DSP/BIOS - G53**

In this lesson, you modify the example you worked on in the previous lesson.

**Note:** If you did not complete the previous lesson, you can copy example files from the volume3 folder that reflect the state of the example at the end of the previous lesson. Copy all the files and folders from C:\CCStudio\_v3.10\tutorial\target\Volume2 to C:\CCStudio\_v3.10\myprojects\Volume2.

1. Copy only the following files from the C:\CCStudio\_v3.10\tutorial\target\Volume4\ folder to the C:\CCStudio\_v3.10\myprojects\Volume2\ folder that you used in the previous lesson. (Note that you should not copy all the files from the volume4 folder. In particular, do not copy the volume.cdb file.) Click Yes when you are asked whether to replace the existing

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007

**Getting Started With the Code Composer Studio Tutorial**

Page 348 sur 548

volume.c file.

- volume.h. This file declares the buffer C-structure as well as define any required constants.
  - volume.c. The source code has been modified to allow you to use the RTDX module to change the load without stopping the target program. You examine the changes in [Reviewing the Source Code Changes](#).
  - loadctrl.exe. This is a simple Windows application written in Visual Basic 5.0. It sends load values to the target in real time using RTDX.
  - loadctrl.frm, loadctrl.frx, loadctrl.vbp. If you have Visual Basic, you can use it to examine these source files for the loadctrl.exe application.
2. If you have not already done so, from the Windows Start menu, choose Programs->Texas Instruments->Code Composer Studio 3.1->Code Composer Studio 3.1. (Or, double-click the Code Composer Studio icon on your desktop.) If another project was already open in Code Composer Studio, right-click on that project's .pjf file in the Project View and choose Close from the shortcut menu.
  3. Choose Project->Open. Select the volume.pjf file in your working folder and click Open.

**Modifying the Configuration File****DSP/BIOS - G53**

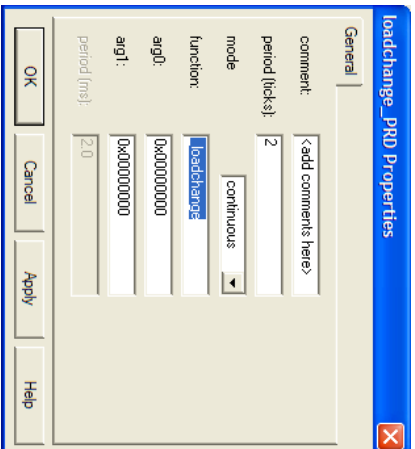
For this lesson, you need to add a periodic function object (PRD) to the configuration file. PRD threads are a special case of SWI threads: they are posted at a periodic rate that may be controlled by the CLK module or by program activity such as data availability. (The volume.cdb file in the C:\CCStudio\_v3.10\tutorial\target\Volume4\ folder already contains this object.)

1. In the Project View, double-click on the volume.cdb file to open it.
2. Click the + sign next to the Instrumentation and Scheduling categories to display their lists of modules.
3. In the Properties dialog for the LOG\_system object, change the buflen [property](#) to 512 words, and click OK.
4. Right-click on the PRD - Periodic Function Manager and choose Insert PRD from the context menu.
5. [Rename](#) the PRD0 object to loadchange\_PRD.
6. Right-click on the loadchange\_PRD object and choose Properties from the context menu.
7. Change the period (ticks) to 2. By default, the PRD manager uses the CLK manager to drive PRD execution. The default properties for the CLK class make a clock interrupt trigger a PRD tick each millisecond. So, this PRD object runs its function every 2 milliseconds.

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007

8. Change the function to `_loadchange`. This PRD object executes the `loadchange_C` function each time the period you chose elapses. (Recall that you need to use an underscore prefix for C functions in the configuration file.) You look at this function in the next section. Click OK.



9. Notice that a SWI object called `PRD_swi` was created automatically when you added the `loadchange_PRD` object. This may not appear until you save the configuration. This software interrupt executes periodic functions in real time. Therefore, all PRD functions are called within the context of a software interrupt and can yield to hardware interrupts. In contrast, CLK functions run in the context of a hardware interrupt. (The `KNL_swi` object runs a function that runs the TSK manager. To learn more about tasks, see the [Using Tasks for Blocking and Yielding](#) lesson.)
10. Right-click on the PRD - Periodic Function Manager and choose Insert PRD from the context menu.
11. Rename the PRD0 object to `calcStartupLoad`.
12. Right-click on the `calcStartupLoad` object and choose Properties from the context menu.
13. Change the period (ticks) to 6000.
14. Change the mode to one-shot.
15. Change the function to `_startupLoadCalc`. Click OK.
16. Click the + sign next to the CLK -Clock Manager. Notice that the CLK object called `PRD_clock` runs a function called `PRD_F_tick`. This function causes the DSP/BIOS system clock to tick (by calling the `PRD_tick_API` function) and the `PRD_swi` software interrupt to be posted if any PRD functions need to run. `PRD_swi` runs the functions for all the PRD objects whose period has elapsed.

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007

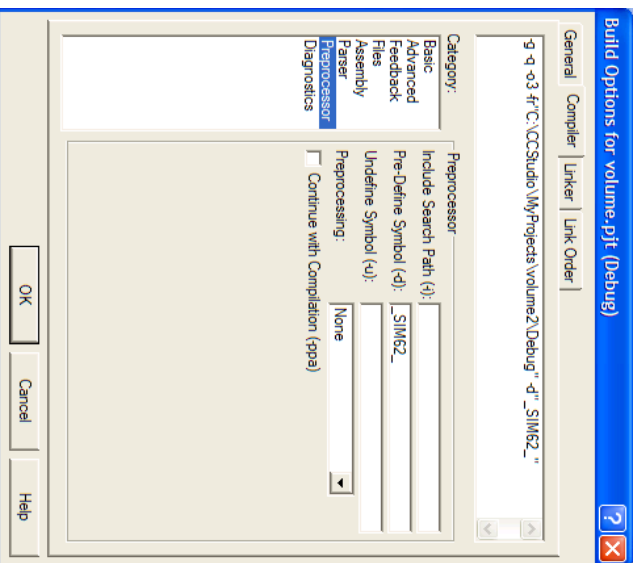
Getting Started With the Code Composer Studio Tutorial


Page 350 sur 548

17. Right click on the PRD Manager, and choose Properties from the context menu. The PRD manager has a property called Use CLK Manager to drive PRD. Make sure this box is checked for this example. In your own projects, if you remove the check mark from this box, the PRD\_clock object would be deleted automatically. Your program could then call `PRD_tick` from some other event, such as a hardware interrupt, to drive periodic functions.
18. Recall that the processing `_SWI` object has a mailbox value of 10. That mailbox value is decremented by the `dataIO_CLK` object, which runs the `dataIO_C` function every millisecond. As a result, the processing `_SWI` runs its function every 10 milliseconds. In contrast, the `loadchange_PRD` object should run its function every 2 milliseconds.
19. Choose File->Close. You are asked whether you want to save your changes to `volume.cdb`. Click Yes. Saving this file also generates the following files: `volumecfg.cmd`, `volumecfg.h`, `volumecfg.i62`, `volumecfg.s62`, and `volumecfg.c.c`.
20. From the Project menu, choose Build Options.
21. In the Build Options dialog window, click on the Compiler tab, then on Preprocessor in the Category list.
22. Type `_SIM62_` in the Pre-Define Symbol (-d) field. Your build options should look like this:

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007



23. Click OK to save the changes.
24. Choose Project→Rebuild All or click the  (Rebuild All) toolbar button.



### Reviewing the Source Code Changes

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007

### Getting Started With the Code Composer Studio Tutorial

Page 352 sur 548

DSP/BIOS - 653

Double-click on the volume.c file in the Project View to see the source code in the right half of the Code Composer Studio window.

Since you copied the volume.c file from the C:\CCStudio\_3.10\tutorial\target\Volume4\ folder to your working folder, the source code now contains the following differences from the source code used at the end of the [previous lesson](#):

- Added the following to the list of included header files:
 

```
#include <rdx.h>
```
- Added the following to the global declarations:
 

```
RTDX_CreateInputChannel(control_channel);
```
- Added the following to the function declarations:
 

```
Void loadchange(Void);
```
- Added the following call to the main function:
 

```
RTDX_enableInput(&control_channel);
```
- The following function is called by the PRD object you created in [Modifying the Configuration File](#). This is where the processor is controlled.
 

```
/* ===== loadchange =====
 * FUNCTION: Called from loadchange_PRD to periodically update the load value.
 * PARAMETERS: none.
 * RETURN VALUE: none.
 */
Void loadchange()
{
    static Int control = MINCONTROL;

    /* Read new load control when host sends it */
    if (RTDX_ChannelBusy(&control_channel)) {
        RTDX_ReadNB(&control_channel, &control, sizeof(control));
        if ((control < MINCONTROL) || (control > MAXCONTROL)) {
            LOG_printf(trace,"Control value out of range");
        }
        else {
            if(control) {
                processing_load = getLoadFactor(control);
            }
        }
    }
}
```

```
/* ===== loadchange =====
 * FUNCTION: Called from loadchange_PRD to periodically update the load value.
 * PARAMETERS: none.
 * RETURN VALUE: none.
 */
Void loadchange()
{
    static Int control = MINCONTROL;

    /* Read new load control when host sends it */
    if (RTDX_ChannelBusy(&control_channel)) {
        RTDX_ReadNB(&control_channel, &control, sizeof(control));
        if ((control < MINCONTROL) || (control > MAXCONTROL)) {
            LOG_printf(trace,"Control value out of range");
        }
        else {
            if(control) {
                processing_load = getLoadFactor(control);
            }
        }
    }
}
```

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007



```
else {
    processingLoad = control;
}
#endif_28
LOG_printf(&trace,"Load value = %d", (Avg)control);
#else
LOG_printf(&trace,"Load value = %d", control);
#endif
}
}
```

This function uses RTDX API functions to change the load of the processing signal in real time. Notice the following aspects of these changes:

- The call to RTDX\_enableInput enables the input channel called control\_channel so that data can flow on it from the host to the target. When you run the program, a Visual Basic host client writes a load control value on that channel, thereby sending it to the target application.
- The call to RTDX\_readNB asks the host to send a load control value on the control\_channel and stores the result in the variable called control. This call is non-blocking. It returns without waiting for the host to send the data. The data is delivered when the host client writes to control\_channel. From the time of the call to RTDX\_readNB until the data is written to the variable control, this channel is busy, and no additional requests can be posted on this channel (that is, calls to RTDX\_readNB do not succeed). During that time, the call to RTDX\_channelBusy returns TRUE for control\_channel.
- The processingLoad = control; statement sets the processing load to the value specified by the control.



### Using the RTDX Control to Change the Load

#### DSP/BIOS - G53

You could test this program by putting a Probe Point within the processing function and viewing graphs of input and output data. (To see how, go to the [Data Visualization](#) lesson in the Code Composer Studio IDE module of the tutorial.)

However, in order to focus on the real-time aspects of DSP/BIOS, let us assume that you have already tested the signal processing algorithm. At this stage of development, your focus should be on making sure threads can meet their real-time deadlines. In addition, Probe Points halt the target and can interfere with the real-time aspects of the test.

1. Choose File→Load Program. In the Debug subfolder, select the program you just rebuilt, volume.out, and click Open.
2. Choose DSP/BIOS→RTA Control Panel.
3. Verify that the Enable SW1 logging, Enable PRD logging, Enable CLK logging, Enable SWI accumulators, and Enable PRD accumulators boxes are checked. Scroll down and verify that

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

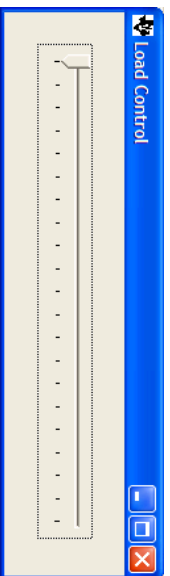
26/09/2007

### Getting Started With the Code Composer Studio Tutorial

Page 354 sur 548

the Global host enable box is checked.

4. Choose DSP/BIOS→Execution Graph. *Resize* the Execution Graph window as needed.
5. Choose DSP/BIOS→Statistics View.
6. Right-click on the Statistics View area and choose Property Page from the context menu. In the General tab, verify that loadchange\_PRD, PRD\_sw1, and processing\_SW1 are enabled.
7. In the Units tab of the Statistics View Properties window, select the loadchange\_PRD object and type "ticks" as the unit label. Click OK.
8. Resize the Statistics View window and its columns so that you can see the Count, Max, and Average columns.
9. Right-click on the RTA Control Panel and choose Property Page from the context menu.
10. Set the Refresh Rate for Message Log/Execution Graph to 1 second and the Refresh Rate for Statistics View/CPU Load Graph to 0.5 seconds. Then click OK.
11. Choose Tools→RTDX→Configuration Control.
12. Notice that RTDX is already enabled. This happened behind-the-scenes in the [Modifying the Configuration File](#) lesson when you opened a DSP/BIOS control. DSP/BIOS controls configure and enable RTDX in continuous mode. In continuous mode, RTDX does not record data received from the target in a log file (as it does in non-continuous mode). This allows continuous data flow. (If your program does not use DSP/BIOS, you can use the RTDX control to configure and enable RTDX directly.)
13. Using Windows Explorer, run loadctrl.exe, which is located in your working folder (usually C:\CCStudio\_v3.10\myprojects\volume2). The Load Control window appears.



Each mark on the slider changes the load value by 50, or about 50,000 instructions.

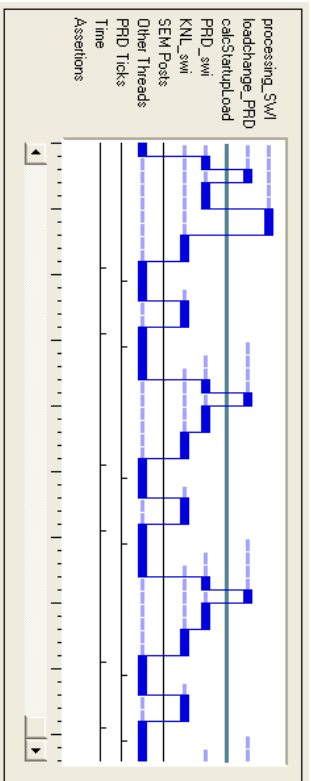
This simple Windows application was written using Visual Basic and RTDX. If you have Visual Basic, you can examine the source files for the loadctrl.exe application stored in the C:\CCStudio\_v3.10\Tutorial\Target\_Volume4 folder. This application uses the following RTDX functions:

- rdx.Open("control\_channel", "W"). Opens a control channel to write information to the target when you open the application
- rdx.Close(). Closes the control channel when you close the application
- rdx.Write12(data12, bufstate). Writes the current value of the slider control to control\_channel so that the target program can read this value and use it to update the load

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007

14. Choose Debug→Run or click the  (Run) toolbar button.



Notice that processing\_SWI occurs once every 10 time ticks (and PRD ticks). PRD\_SWI runs once every 2 PRD ticks. The loadchange\_PRD runs within the context of PRD\_SWI. These are the expected execution frequencies.

PRD statistics are measured in PRD ticks. SWI statistics are measured in instruction cycles. The Max field for loadchange\_PRD shows that there is less than a full PRD tick between the time this function needs to start running and its completion. (The statistics for the other objects vary depending on your DSP platform.)

Statistics View	Count	Total	Max	Average
STS		0 ticks	0 ticks	0
loadchange_PRD	503	0	-2.14748e+009	0
calcStartupLoad	0	0	1308 inst	1146.86
PRD_SWI	503	576872 inst	1496.48 inst	1496.48
processing_SWI	100	1496.48 inst	1500 inst	0.00
TSK_idle	0	0 inst	-21.47483648 inst	0.00
IDL_busyDdi	6698	-435283	-63	-64.987
processingLoad_STS	100	2558	27	26.58

15. Use the Load Control window to gradually increase the processing load. (If you move the slider in the Load Control window while the DSP program is halted, the new load control values are buffered on the host by RTDX. These have no effect until the DSP application runs again and calls RTDX\_readNB to request updated load values from the host.)
16. Repeat step 15 until you see the Max and Average values for loadchange\_PRD increase and red squares appear in the Assertions row of the Execution Graph. Assertions indicate that a thread is not meeting its real-time deadline. (Again, the actual statistics vary depending on your DSP platform.)

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007

Getting Started With the Code Composer Studio Tutorial

Page 356 sur 548

What is happening? The Max value for loadchange\_PRD increases when you increase the load beyond a certain point. With the increased load, the processing\_SWI takes so long to run that the loadchange\_PRD cannot begin running until long past its real-time deadline.

When you increase the load so much that the low-priority idle loop is no longer executed, the host stops receiving real-time analysis data and the DSP/BIOS analysis tools stop updating. This is called "starving the idle loop." Halting the target updates the analysis tools with the queued data.



### Modifying Software Interrupt Priorities

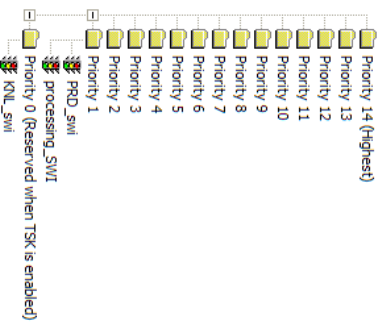
DSP/BIOS - G53

To understand why the program is not meeting its real-time deadline, you need to examine the priorities of the software interrupt threads.

1. Select Debug→Halt to halt the target.
2. In the Project View, double-click on the volume.cdb file to open it.
3. Click the + sign next to the Scheduling category to display its list of modules.
4. Highlight the SWI - Software Interrupt Manager. Notice the SWI object priorities shown in the right half of the window. (The KNL\_SWI object runs a function that runs the TSK manager. This object must always have the lowest SWI priority. Tasks are not used in this lesson.)

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007



Because the PRD\_sw1 and processing\_sw1 objects both have the same priority level, the PRD\_sw1 cannot preempt the processing\_sw1 while it is running.

The processing\_sw1 needs to run once every 10 milliseconds and PRD\_sw1 needs to run every 2 milliseconds. When the load is high, processing\_sw1 takes longer than 2 milliseconds to run and prevents PRD\_sw1 from meeting its real-time deadline.

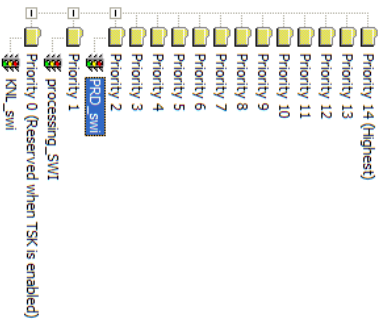
- To correct this problem, use your mouse to select and drag PRD\_sw1 to a higher priority level, such as Priority 2.


file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007

Getting Started With the Code Composer Studio Tutorial

Page 358 sur 548



- Select File→Save to save your changes.
- Select File→Close to close volume.cdb.
- Choose Project→Build or click the  (Incremental Build) toolbar button.
- Select File→Reload Program.
- Select Debug→Run to run the example again. Use the RTDX-enabled Windows application loader.exe to change the load (as in Using the RTDX Control to Change the Load).
- Notice that you can now increase the load without causing PRD\_sw1 to miss its real-time deadline.

**Note:** It is still possible to starve the idle loop by increasing the processing load to its maximum value. When you increase the load so much that the CPU no longer has time to run the low-priority idle loop, the host stops receiving real-time analysis data and the DSP/BIOS analysis tools stop updating.

- Before continuing to the next lesson (after completing [Things to Try](#)), perform the following steps to prepare for it.
- Click Debug→Halt or press Shift F5 to stop the program.
- Close all GEL dialogs, DSP/BIOS analysis tools, and source windows

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007



---

## Things to Try

---

**DSP/BIOS - GS3**

To further explore DSP/BIOS, try the following:

- When you increase the load, the Execution Graph shows that processing\_SW1 requires more than one PRD tick to run. Does this mean that processing\_SW1 is missing its real-time deadline? [Answer](#)
- What would happen if the processing function were called directly from a hardware interrupt rather than being deferred to a software interrupt? [Answer](#)
- View the CPU Load Graph. Use the RTA Control Panel to turn the statistics accumulators on and off. Notice that the CPU Load Graph appears unaffected. This demonstrates that the statistics accumulators place a very small load on the processor.  
  
How much do the statistics accumulators affect the statistics for processing\_SW1? Watch the Statistics View as you turn the statistics accumulators on and off. The difference is a precise measurement of the number of instructions each accumulator requires. Remember to right-click and clear the Statistics View to see the effect. [Answer](#)
- Add calls to STS\_set and STS\_delta in the loadchange function like the ones you added in the [Adding Explicit STS Instrumentation](#) lesson in the Debugging Program Behavior tutorial. How does this change affect the CPU load? Now, add calls to STS\_set and STS\_delta in the dataIO function. How does this change affect the CPU load? Why? Consider the frequency at which each function is executed. [Answer](#)

This concludes the Analyzing Real-Time Behavior lesson. The next lesson will cover connecting to virtual I/O devices.



---

## Overview

---

**DSP/BIOS - GS4**

This lesson shows how to connect a program to an I/O device. You use the RTDX, HST, and PIP modules of the DSP/BIOS API. The first program variation uses RTDX for signal transfer and volume control. The second variation uses DSP/BIOS pipes for signal transfer and RTDX for volume control.

**Learning Objectives:**

- Modify the project for DSP/BIOS

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007

---

## Getting Started With the Code Composer Studio Tutorial

---

Page 360 sur 548

- Use DSP/BIOS to connect a program to an I/O device
- Use RTDX and DSP/BIOS pipes to modify the project
- Use DSP/BIOS tools to measure performance

**Examples used in this lesson: hostio1, hostio2****Application Objective:**

The example used in this lesson simulates a DSP application that digitizes an audio signal, adjusts its volume, and produces an analog output at the adjusted volume. For simplicity, no actual device is used to send and receive analog data in this example. Instead, the example tests the algorithm using host-generated digital data.

This lesson consists of the following steps. To go directly to a step, click on the link below:

- [Opening and Examining the Project](#)
- [Reviewing the C Source Code](#)
- [Reviewing the Signalprog Application](#)
- [Running the Application](#)
- [Modifying the Source Code](#)
- [Adding Channels and Interrupts to the Configuration File](#)
- [Running the Modified Program](#)
- [Things to Try](#)



The forward arrow will take you to the next page in this lesson.

---

## Opening and Examining the Project

---

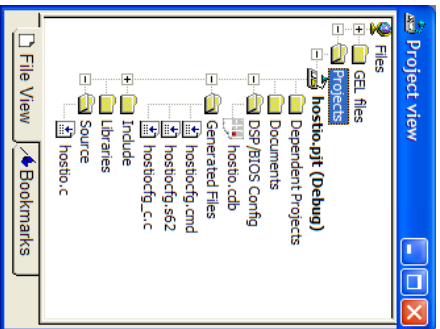
**DSP/BIOS - GS4**

You begin by opening a project with Code Composer Studio and examining the source code files and libraries used in that project.

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007

1. Create a folder called `hostio` in the `C:\CCStudio_v3.10\myprojects` folder.
2. Copy all the files and folders from the `C:\CCStudio_v3.10\tutorial\target\hostio1` folder to this new folder.
3. If you have not already done so, from the Windows Start menu, choose Programs->Texas Instruments->Code Composer Studio 3.1. (Or, double-click the Code Composer Studio icon on your desktop.) If another project was already open in Code Composer Studio, right-click on that project's .jpt file in the Project View and choose Close from the shortcut menu.
4. Choose Project->Open. Select the `hostio.jpt` file in the folder you created and click Open.
5. Expand the Project View by clicking the + signs next to Projects, `hostio.jpt`, DSP/BIOS Configuration, Generated Files, and Source. The `hostiocfg.cmd` file, which was created when the configuration was saved, includes a large number of DSP/BIOS header files. You do not need to examine all these header files.



6. The files used in this project include:
  - `hostio.c`. This is the source code for the main program. You examine the source code in the next section.
  - `signalprog.exe`. This Visual Basic application generates a sine wave and displays the input and output signals.
  - `slider.exe`. This Visual Basic application allows you to control the volume of the output signal.

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007

## Getting Started With the Code Composer Studio Tutorial

Page 362 sur 548

- `hostioCfg.cmd`. This linker command file is created when you save the configuration file. The only object that has been added to the default configuration is a LOG object called trace.
- `hostioCfg.h`. This header file is created when you save the configuration file. It includes DSP/BIOS module header files and declares external variables for objects created in the configuration file.
- `hostioCfg.s62`. This assembly file is created when you save the configuration file and contains DSP/BIOS configuration settings.
- `hostioCfg.h62`. This header file is created when you save the configuration file and is included by `volumeCfg.s62`.
- `hostioCfg.c.c`. This file is created when you save the configuration file. It contains code for Chip Support Library (CSL) structures and settings.



## Reviewing the C Source Code

DSP/BIOS - GS4

The example in this lesson simulates a DSP application that digitizes an audio signal, adjusts its volume, and produces an analog output at the adjusted volume.

For simplicity, no actual device is used to send and receive analog data in this example. Instead, the example tests the algorithm using host-generated digital data. Input and output data and volume control are transferred between the host and the target using RTDX.

A Visual Basic application running on the host uses RTDX to generate the input signal and display the input and output signals. This application allows developers to test the algorithm without stopping the target. Similar methods can be used to create display controls for real-time testing of other applications. You examine the Visual Basic application in [Reviewing the Signalprog Application](#).

1. Double-click on the `hostio.c` file in the Project View to see the [source code](#).
2. Notice the following aspects of this example:
  - Three RTDX channels are declared globally. The first input channel controls the volume. The second input channel receives the input signal from the host. The output channel sends the output signal from the target to the host. (Input and output channels are named from the perspective of the target application; input channels receive data from the host, and output channels send data to the host.)
  - The call to `RTDX_channelBusy` returns FALSE if the channel is not currently waiting for input. This indicates that the data has arrived and can be read. As in the previous lesson, the call to `RTDX_readNB` is non-blocking; it returns control to the DSP application without waiting to receive the data from the host. The data is delivered asynchronously when the host client writes it to the control channel.
  - Calls to `RTDX_Poll` are used to communicate with the underlying RTDX layer to read and write data.
  - The call to `RTDX_read` waits for data if the channel is enabled.

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007

- The call to `RTDX_write` writes the contents of the buffer to the output `RTDX_channel` if the channel is enabled.
- While control\_channel is enabled by the target via a call to `RTDX_enableInput`, the other `RTDX_channel`s are not enabled from this program. Instead, a host program described in the next section enables these channels. This is because the slider control, which uses the `control_channel`, is viewed as an integral part of the application. By enabling this channel in the target program, you know the channel is enabled while the application is running. In contrast, the `A2D` and `D2A` channels are used to test the algorithm. Hence, these channels are enabled and disabled by the host application.



### Reviewing the Signalprog Application

DSP/BIOS - GS4

The source code for the Visual Basic (version 5) `signalprog.exe` application is available in the `signalfrm.frn` file. Details about this application are provided in the `signalprog.pdf` Adobe Acrobat file. In this section, you examine a few of the routines and functions that are important for this example.

- `Test_ON`: This routine runs when you click the `Test_ON` button. It creates instances of the `RTDX` exported interface for the input channel (`toDSP`) and for the output channel (`fromDSP`). Then it opens and enables both of these channels. The channels in the `signalprog.exe` application are the same channels declared globally in the `hostio.c` source code.

This routine also clears the graphs and starts the timer used to call the `Transmit_Signal` and `Receive_Signal` functions.

These global declarations made earlier in the Visual Basic source code connected the `READ_CHANNEL` and `WRITE_CHANNEL` used in the `Test_ON` routine to the `D2A_channel` and `A2D_channel` used in `hostio.c`:

```
' Channel name constants
Const READ_CHANNEL = "D2A_channel"
Const WRITE_CHANNEL = "A2D_channel"
```

- `Test_OFF`: This routine disables, closes, and releases the `RTDX` objects created by `Test_ON`. It also disables the timer.
- `Transmit_Signal`: This function generates a sine wave signal and displays it in the `Transmitted_Signal` graph. Then the function attempts to transmit the signal to the target using the `Write` method of the `toDSP_channel`.
- `Receive_signal`: This function uses the `ReadSAI2` method of the `fromDSP_channel` to read a signal from the target. It displays the signal in the `Received_Signal` graph.
- `tmr_MethodDispatch_Timer`: This routine calls the `Transmit_Signal` and `Receive_Signal` functions. This routine is called at 1 millisecond intervals after the timer object is enabled by the `Test_ON` routine.

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007



Getting Started With the Code Composer Studio Tutorial

Page 364 sur 548



### Running the Application

DSP/BIOS - GS4

1. Choose `Project`→`Build` or click the  (Incremental Build) toolbar button.
2. Choose `File`→`Load Program`. In the `Debug` subfolder, select `hostio.out` and click `Open`.
3. Choose `Tools`→`RTDX`→`Configuration Control`.
4. If the `Configure` button is dimmed, remove the checkmark from the `Enable RTDX` checkbox.
5. Click `Configure` in the `RTDX Configuration Control` window. In the `RTDX Configuration Page`, select `Continuous mode`. Then, click `OK`.
6. In the `RTDX Configuration Control` window, place a checkmark in the `Enable RTDX` checkbox. Then right-click on the `RTDX Configuration Control` window and select `Hide`.
7. Choose `DSP/BIOS`→`Message Log`. Select `Trace` in the `Log Name` list.
8. Choose `Debug`→`Run` or click the  (Run) toolbar button.
9. Using `Windows Explorer`, click on `slider.exe` to run the program in the folder for `hostio`.
10. The `Select Board And Processor` dialog window will appear; select your target board and processor name from the list. Then click `OK`. You see the following `Visual Basic` application.

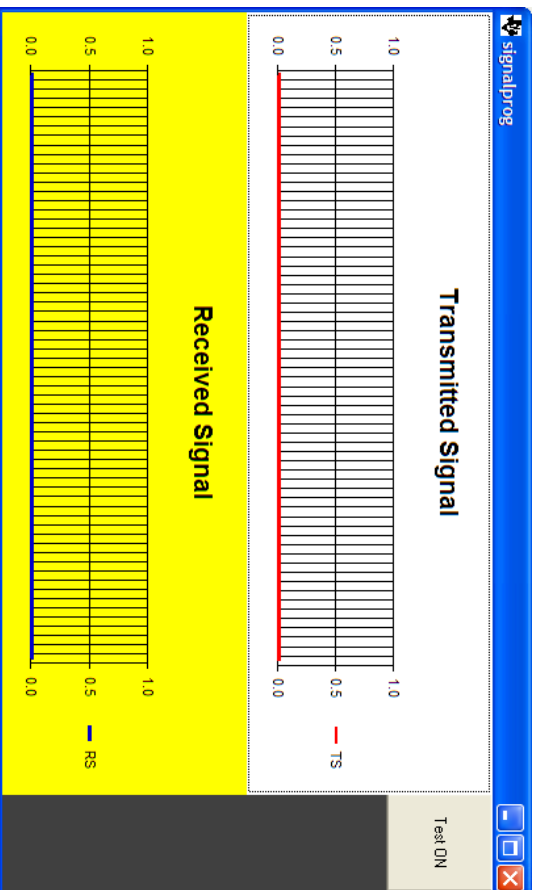


The slider.exe program must be started after `RTDX` is enabled and the program is running because it creates and opens the `RTDX` control channel when you run the program. If `RTDX` is not enabled at this point, `slider.exe` cannot open the channel.

11. Using `Windows Explorer`, click on `signalprog.exe` to run the program in the folder for `hostio`. You see the following `Visual Basic` application.

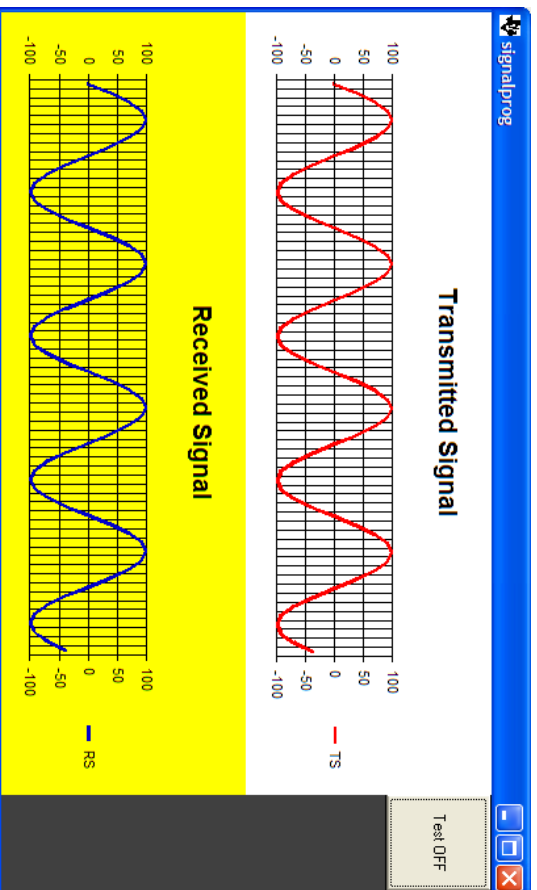
file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007



The signalprog.exe program can be started at any point. It does not use RTDX until you click the Test On button.

12. Click Test On in the signalprog window.
13. The Select Board And Processor dialog window will appear; select your target board and processor name from the list. Then click OK. After a few seconds, you see the transmitted signal and received signal in the Signalprog window.



14. Slide the control in the Volume Slider window. This changes the volume of the output signal. Watch the amplitude of the Received Signal graph change. (Only the scale values to the left and right of the graph change. The graph changes scale automatically to accommodate the size of the sine wave and the size of the window.)

**Note:** The initial setting of the Volume Slider bar is not synchronized with the application. They are synchronized the first time you move the slider bar.

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

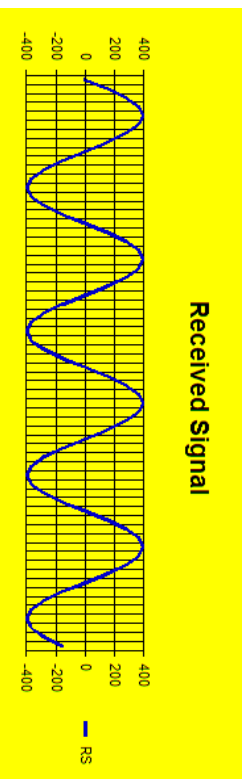
26/09/2007

Getting Started With the Code Composer Studio Tutorial

Page 366 sur 548

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007



15. Close the Volume Slider application. This stops the input and output channels.
16. Click Test Off in the signalprog window. This closes the control channel.
17. Close the signalprog application window.
18. Click Debug →Halt or press Shift F5 to stop the program.
19. You now see the "hostio example started" message from the call to LOG\_print in the Message Log area. You did not see this message earlier because the entire program runs within the main function. DSP/BIOS communicates with the host PC within the idle loop. Until a program returns from main, it never enters the idle loop. Therefore, if you want to see the effects of DSP/BIOS calls when the program runs, your program should perform its functions after returning from main. The modified version of hostio.c used in the next section shows this technique.



### Modifying the Source Code

DSP/BIOS - GS4

Now you modify the example to use the host channels and pipes provided with DSP/BIOS. The modified example still tests your DSP algorithm in real time. Rather than generating a sine wave on the host, this time the data comes from a host file.

The HST module provides a more direct path toward implementing I/O with peripheral devices. The HST module uses the PIP module for host I/O. You can use the PIP module API with minimal modifications to the source code once the I/O devices and hardware interrupt functions are ready for testing.

1. Copy only the following files from the `C:\CCStudio_v3.10\tutorial\target\hostio2\` folder to your working folder, which is usually `C:\CCStudio_v3.10\myprojects\hostio`. (Note that you should not copy all the files from the hostio2 folder. In particular, do not copy the hostio.cdb file.) Click Yes when you are asked whether to replace the existing hostio.c file.

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007

### Getting Started With the Code Composer Studio Tutorial

Page 368 sur 548

- hostio.c. The source code has been modified to use the HST and PIP modules of the DSP/BIOS API instead of RTDX to transfer the input and output signals
  - input.dat. This file contains input data
2. You may get a message from CCSStudio prompting you to reload hostio.c, click on Yes to reload the program.
  3. Double-click on the hostio.c file in the Project View to see the **source code** in the right half of the Code Composer Studio window. The source code now contains the following differences from the source code used earlier in this lesson:
    - Added the following to the list of included header files:
 

```
#include <str.h>
#include <pip.h>
```
    - Removed the BUFSIZE definition, the global declarations of `inp_buffer` and `out_buffer`, and the RTDX input and output channel declarations. This example retains the RTDX channel used to control the volume.
    - Moved the input and output functionality from a while loop in the main function to the `A2Dscaled2A` function.
- The `A2Dscaled2A` function is called by the `A2Dscaled2A_SWI` object. You create this SWI object in the next section and make it call the `A2Dscaled2A` function.
- The `A2Dscaled2A_SWI` object passes two HST objects to this function. This function then calls `HST_getpipe` to get the address of the internal PIP object used by each HST object. Calls to `PIP_getReaderNumFrames` and `PIP_getWriterNumFrames` then determine whether there is at least one frame in the input pipe that is ready to be read and one frame in the output pipe that can be written to.
- Using the same RTDX calls used in [Reviewing the C Source Code](#), the function gets the volume setting from the RTDX control channel.
- The call to `PIP_get` gets a full frame from the input pipe. The call to `PIP_getReaderAddr` gets a pointer to the beginning of the data in the input pipe frame and `PIP_getReaderSize` gets the number of words in the input pipe frame.
- The call to `PIP_alloc` gets an empty frame from the output pipe. The call to `PIP_getWriterAddr` gets a pointer to the location to begin writing data to in the output pipe frame. The function then multiplies the input signal by the volume and writes the results to the frame using the pointer provided by `PIP_getWriterAddr`.
- The call to `PIP_put` puts the full frame into the output pipe. The call to `PIP_free` recycles the input frame so that it can be reused the next time this function runs.
- Added an error function, which writes an error message to the trace log and then puts the program in an infinite loop. This function runs if `A2Dscaled2A` runs when there are no frames of data available for processing.

[More about Host Channels and Pipes](#)

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007





### Adding Channels and Interrupts to the Configuration File

DSP / BIOS - G54

The `AZDscaled2A` function is called by an SWI object and uses two HST objects. You create these objects in this section. (The `hostio.cdb` file in the `C:\CCStudio_v3.1.0\tutorial\target\hostio2\` folder already contains these objects.)

The `AZDscaled2A` function also references two PIP objects, but these objects are created internally when you create the HST objects. The `HST_getpipe` function gets the address of the internal PIP object that corresponds to each HST object.

1. In the Project View, double-click on the `hostio.cdb` file to open it.
2. Click the + sign next to the Scheduling and INPUT/OUTPUT categories to display their lists of modules.
3. Right-click on the HST - Host Channel Manager and choose Insert HST.

Notice that there are HST objects called `RTA_fromHost` and `RTA_loHost`. These objects are used internally to update the DSP/BIOS Analysis Tools.

4. Rename the new HST0 object to `input_HST`.
5. Right-click on the HST - Host Channel Manager and choose Insert HST.
6. Rename the new HST0 object to `output_HST`.

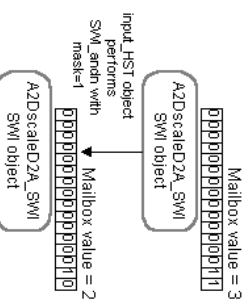
7. Set the following **properties** for these objects: First set the properties in the `input_HST` column for the `input_HST` object. Then, set the properties in the `output_HST` column for the `output_HST` object. The default settings for other properties are correct. The properties mode and framesize are both under the General tab, while `notify`, `arg0`, and `arg1` are under the Notify Function tab.

Property	input_HST	output_HST
mode	Input	output
framesize	64	64
notify	_SWI_andnHook	_SWI_andnHook
arg0	AZDscaled2A_SWI	AZDscaled2A_SWI
arg1	1	2

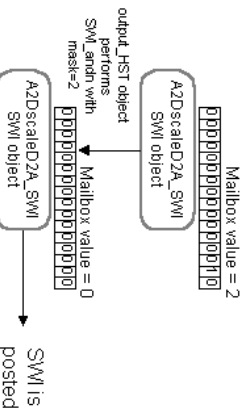
These properties have the following effects:

- mode. This property determines which ends of the pipe are managed by the target program and by the Host Channel Control. An input channel sends data from the host to the target. An output channel sends data from the target to the host.
- framesize. This property sets the size of each frame in the channel. Use 64 words, which is the same value as the `BUFSIZE` defined in the [Reviewing the C Source Code](#) lesson.
- notify, arg0, arg1. These properties specify the function to run when this input channel contains a full frame of data and the arguments to pass to that function. The `_SWI_andnHook` function provides another way to manipulate a SWI object's mailbox.

In the [Reviewing the C Source Code](#) lesson, you used the `SWI_dec` function to decrement the mailbox value and run the SWI object's function when the mailbox value reached zero. The `SWI_andnHook` function treats the mailbox value as a bitmask. It clears the bits specified by the second argument passed to the function. The SWI object is posted when the mailbox value becomes zero. So, when this channel contains a full frame (because the target filled a frame), it calls `SWI_andnHook` for the `AZDscaled2A_SWI` object and causes it to clear the smallest bit of the mailbox.



When this output channel contains an empty frame (because the target read and released a frame), it uses `SWI_andnHook` to clear the second smallest bit of the mailbox.



(Use the `SWI_andnHook` function if you are specifying the function within the Configuration Tool. If you are calling this function from a source file, use the `SWI_andn` function.)

### Getting Started With the Code Composer Studio Tutorial

Page 370 sur 548

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007

8. Right-click on the SW1 - Software Interrupt Manager and choose Insert SW1.
9. Rename the new SW10 object to A2Dscaled2A\_SW1.
10. Set the following [properties](#) for A2Dscaled2A\_SW1 and click OK.

**A2Dscaled2A\_SW1 Properties**

General

comment: <add comments here>

function: A2Dscaled2A

priority: 1

mailbox: 3

arg0: Input\_HST

arg1: output\_HST

OK Cancel Apply Help

- function. This property causes the object to call the A2Dscaled2A function when this software interrupt is posted and runs.
  - mailbox. This is the initial value of the mailbox for this object. The input\_HST object clears the first bit of the mask and the output\_HST object clears the second bit of the mask. When this object runs the A2Dscaled2A function, the mailbox value is reset to 3.
  - arg0, arg1. The names of the two HST objects are passed to the A2Dscaled2A function.
11. Choose File→Close. You are asked whether you want to save your changes to hostio.cdb. Click Yes. Saving the configuration also generates the following files: hostiocfg.cmd, hostiocfg.h, hostiocfg.h62, hostiocfg.s62, and hostiocfg.c.c.



### Running the Modified Program



**DSP/BIOS - GS4**

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007

### Getting Started With the Code Composer Studio Tutorial

Page 372 sur 548

1. Choose Project→Rebuild All or click the  (Rebuild All) toolbar button.
2. Choose File→Load Program. In the Debug subfolder, select the program you just rebuilt, hostio.out, and click Open.
3. Choose DSP/BIOS→Host Channel Control. The Host Channel Control lists the HST objects and allows you to bind them to files on the host PC and to start and stop the channels.
4. Choose Debug→Run or click the  (Run) toolbar button.

**Host Channel Control**

Channel	Transferred	Limit	State	Mode	Binding	Repeat
input_HST	0B	0KB	Unbound	Input	<unbound>	<unbound>
output_HST	0B	0KB	Unbound	Output	<unbound>	<unbound>

5. Right-click on the input\_HST channel and choose Bind from the context menu.
6. Select the input.dat file in your working folder (usually C:\CCStudio\_v3.10\myprojects\hostio) and click Bind.
7. Right-click on the output\_HST channel and choose Bind from the context menu.
8. Type output.dat in the File Name box and click Bind.

**Host Channel Control**

Channel	Transferred	Limit	State	Mode	Binding	Repeat
input_HST	0B	0KB	Stopped	Input	C:\CCStudio...	C:\CCStudio...
output_HST	0B	0KB	Stopped	Output	C:\CCStudio...	C:\CCStudio...

9. Right-click on the input\_HST channel and choose Start from the context menu.
10. Right-click on the output\_HST channel and choose Start from the context menu. Notice that the Transferred column shows that data is being transferred.
11. After the data has all been transferred, click Debug→Halt or press Shift F5 to stop the program. The input file is about 500 KB, and the data transfer may take several minutes.

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007

12. In Windows Explorer, right-click on the input.dat file and choose Properties. Do the same for the output.dat file. Notice that the final 18 bytes of the file were not transferred. This is because HST frames are not transferred until they are full. In this example, they must contain 64 bytes in order to be transferred.



---

## Things to Try

**DSP/BIOS - GS4**

Lessons 1 through 4 provide a basic overview of DSP/BIOS. After lesson 4, you can perform the remaining lessons of the DSP/BIOS tutorial on an as-needed basis and in any sequence. You don't need to complete all the lessons. Use the remaining lessons as you encounter DSP/BIOS concepts and modules that you want to understand.

To learn more about DSP/BIOS, see the other topics in this online help system and the DSP/BIOS manuals (which are provided as Adobe Acrobat files).

This concludes the Connecting to Virtual I/O Devices lesson.



---

## Introduction

**DSP/BIOS - PT**

Many real-time DSP applications must perform a number of seemingly unrelated functions at the same time, often in response to external events such as the availability of data or the presence of a control signal. Both the functions performed and when they are performed are important.

These functions are called threads. Different systems define threads either narrowly or broadly. Within DSP/BIOS, the term is defined broadly to include any independent stream of instructions executed by the DSP. A thread is a single point of control that may contain a subroutine, a macro, or a function call.

Multi-threaded programs run on a single processor by allowing higher-priority threads to preempt lower-priority threads and by allowing various types of interaction between threads, including blocking, communication, and synchronization.

DSP/BIOS supports several types of **program threads** with different priorities. Each thread type has different execution and preemption characteristics. The thread types (from highest to lowest priority) are:

- **Hardware Interrupts (HWI)**: includes CLK functions
- **Software Interrupts (SWI)**: includes PRD functions

file:///C:/Documents and Settings/Pierre-Armand/Local Settings/Temp/~hhBA53.htm

26/09/2007

## Getting Started With the Code Composer Studio Tutorial

Page 374 sur 548

- **Tasks (TSK)**
- **Background thread (IDL)**

This section contains the following lessons on using threads:

[Choosing Thread Types](#)

[Scheduling SWI and PRD Threads](#)

[Using Tasks for Blocking and Yielding](#)

### See Also

For additional information about threads, see the following:

- Chapter 4 of the DSP/BIOS User's Guide, which is provided in PDF format
- The DSP/BIOS Modules topic
- Various DSP/BIOS application notes available on the Texas Instruments™ website, including:
  - An Audio Example Using DSP/BIOS (SPRA598)
  - Understanding the Functional Enhancements of DSP/BIOS and Their Utilization in Real-Time DSP Applications (SPRA648)
  - Programming and Debugging Tips for DSP/BIOS (SPRA640)
  - DSP/BIOS Technical Overview (SPRA646)
  - DSP/BIOS by Degrees: Using DSP/BIOS in an Existing Application (SPRA591)
  - Understanding Basic DSP/BIOS Features (SPRA653)



file:///C:/Documents and Settings/Pierre-Armand/Local Settings/Temp/~hhBA53.htm

26/09/2007

## Overview

### DSP/BIOS - PT1

DSP/BIOS supports several types of program threads. Each thread type has different execution and preemption characteristics. The thread types (from highest to lowest priority) are:

- **Hardware interrupts (HWI)**: includes clock (CLK) functions
- **Software interrupts (SWI)**: includes periodic (PRD) functions
- **Tasks (TSK)**
- **Background thread (IDL)**

This lesson helps you decide which type of thread to use for a particular situation. You can use the [thread comparison table](#) or answer the [thread selection questions](#).

Instead of printing this section, it is best to use this section online so that you can find the recommended thread type for your needs by following the links next to your answers.

This lesson consists of the following topics. To go directly to a topic, click on the link below:

[Thread Type Characteristics](#)

[Thread Selection Questions](#)

[About Threads and Multi-Threading](#)



The forward arrow will take you to the next topic in this lesson.

## Thread Type Characteristics

### DSP/BIOS - PT1

This table compares the characteristics of the DSP/BIOS thread types:

	HWI	SWI	TSK	IDL
Priority	highest	second highest	second lowest	lowest
Number of priority levels	DSP dependent	15 PRD function run at priority of PRD_swi SWI object. TSK scheduler runs at lowest SWI priority.	16 (including one for the idle loop)	1
Can block (suspend) own execution	no, runs to completion except for other thread	no, runs to completion except for other thread	Yes	should not; would prevent PC from

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007

## Getting Started With the Code Composer Studio Tutorial

Page 376 sur 548

	preemption	preemption	getting target information
Execution states	inactive, ready, running	inactive, ready, running	ready, running, blocked, terminated
Preemption by thread type disabled by:	HWI_disable, HWI_enter (with certain masks)	SWI_disable	TSK_disable
Posted or made ready to run by:	Interrupt occurs	SWI_post, SWI_andn, SWI_dec, SWI_inc, SWI_or, PRD_tick	TSK_create
Stack used	system stack (1 per program)	system stack (1 per program)	task stack (1 per task)
Context saved when preempted by other thread	customizable	certain registers saved to system stack	entire context saved to task stack
Context saved when blocked	not applicable	not applicable	saves the C register set (see the compiler manual)
Share data with thread via:	streams, queues, pipes, global variables	streams, queues, pipes, global variables	streams, queues, pipes, locks, mailboxes, global variables
Synchronize with thread via:	not applicable	SWI mailbox	mailboxes, global variables, semaphores, mailboxes
Function hooks	no	no	yes: create, delete, exit, task switch, ready
Static creation	Included in default configuration template	Yes	Yes
Dynamic creation	Yes	Yes	Yes
Dynamically change priority	no*	Yes: SWI_raisepri, SWI_restorepri	Yes: TSK_sepri
Implicit logging	none	post and completion events	ready, start, block, resume, and termination events
Implicit statistics	monitored values	execution time	no no none

\* When a HWI function calls HWI\_enter, it can pass a bitmask that indicates which interrupts to enable while the HWI function runs. An enabled interrupt can preempt the HWI function even if the enabled interrupt has a lower priority than the current interrupt.



## Thread Selection Questions

### DSP/BIOS - PT1

Click the answer to this question for a particular thread in your application. Continue answering questions until you reach the topic that recommends a thread type.

Does the thread perform [critical processing](#) and take little time to execute compared to other threads in your application?

- Yes.
- Some of it. Part of the thread performs critical processing. Other parts take longer to execute and are less time-dependent.

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007

- **Not as critical or as fast as other threads.** The thread has a real-time deadline, but the function takes long enough to execute that other threads need to be able to preempt it.
- **No.** The thread performs non-critical background processing, such as collecting and communicating instrumentation data.



### About Threads and Multi-Threading

**DSP/BIOS - PT1**

Many real-time DSP applications must perform a number of functions at the same time, often in response to external events such as the availability of data or the presence of a control signal. Both the functions performed and when they are performed are important. These functions are called threads.

Some operating systems define threads narrowly; others define them broadly. Within DSP/BIOS, the term is defined broadly to include any independent stream of instructions executed by the DSP. A thread is a single point of control that may execute a subroutine, a macro, or a function call.

Traditional programs run from a main function, use a single polling loop, or process events one at a time in the same sequence they occur.

Multi-threaded programs run on a single processor by allowing higher-priority threads to preempt lower-priority threads and by allowing various types of interaction between threads, including blocking, communication, and synchronization.



### Overview

**DSP/BIOS - PT2**

This lesson shows how to manage timing interactions between software interrupts. Then, it shows you how to use Code Composer Studio to detect and fix conflicts that may arise. By following the process of solving timing problems, you learn more about how threads interact.

Learning Objectives:

- Modify the project for DSP/BIOS
- Use DSP/BIOS tools to detect timing problems
- Use SWI and PRD threads to correct scheduling problems

file:///C:/Documents and Settings/Pierre-Armand/Local Settings/Temp/~hhBA53.htm

26/09/2007

### Getting Started With the Code Composer Studio Tutorial

Page 378 sur 548

#### **Examples used in this lesson: echo**

Application Objective:

The example used in this lesson illustrates thread scheduling issues encountered in a common DSP application: a vocoder with echo cancellation. This example intentionally causes a common type of timing conflict.

This lesson consists of the following steps. To go directly to a step, click on the link below:

- [Understanding the Echo Canceled Application](#)
- [Copying and Opening the Project](#)
- [Reviewing the Source Code](#)
- [Configuring Objects to Run the Threads](#)
- [Testing with CCSvStudio](#)
- [Correcting Scheduling Problems, Part 1](#)
- [Correcting Scheduling Problems, Part 2](#)
- [Things to Try](#)



The forward arrow will take you to the next page in this lesson.

### Understanding the Echo Canceled Application

**DSP/BIOS - PT2**

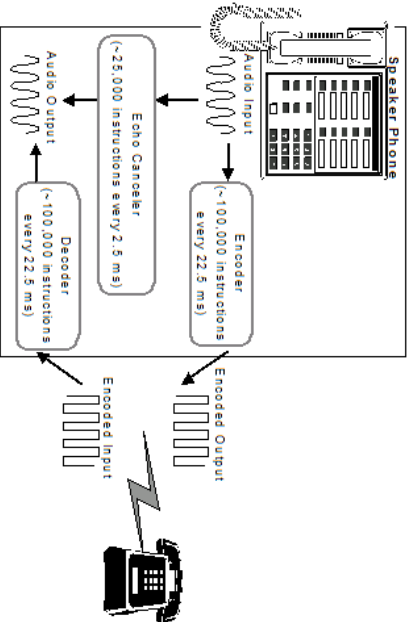
The echo example used in this lesson illustrates thread scheduling issues encountered in a common DSP application: a vocoder with echo cancellation. The echo application consists of three distinct threads:

- An encoder to compress incoming pulse code modulated (PCM) audio data
- A decoder to decompress previously encoded data into PCM audio data
- An echo canceler to suppress output audio levels based on the most recent input audio data (to prevent feedback, for example)

The following figure shows how the example used in this lesson works. The numbers of instructions and timings vary for different applications.

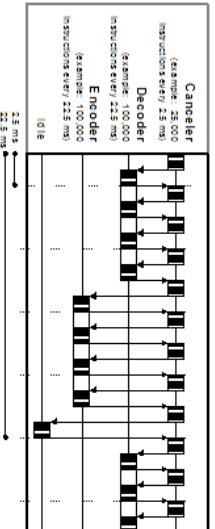
file:///C:/Documents and Settings/Pierre-Armand/Local Settings/Temp/~hhBA53.htm

26/09/2007



Since the echo canceler must react to incoming data in time to prevent feedback, the echo canceler's period is determined by the echo delay between the audio output and the audio input of the system. On the other hand, the encoder and decoder threads have a period determined by the algorithm used to encode and decode the audio data.

To allow these three threads to meet their real-time deadlines, the execution sequence should look similar to this:



To focus on scheduling issues without the complexity of a true audio input/output device, this example uses periodic functions to simulate hardware interrupts that would occur as the result of a frame of audio data becoming available on input, or an empty frame becoming available on output.

Furthermore, the algorithms used in each thread do not do real work; they simply consume a number of CPU cycles determined by global parameters that can be modified in real-time using Code Composer Studio. This allows us to test the system's behavior under varying encoder/decoder loads.

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007

Getting Started With the Code Composer Studio Tutorial

Page 380 sur 548

### Copying and Opening the Project

DSP/BIOS - PT2

Begin by following these steps to copy the project to a working folder and open the project with Code Composer Studio:

1. Create a folder called echo in the C:\CCStudio\_v3.10\myprojects folder.
2. Copy all the files and folders from the C:\CCStudio\_v3.10\tutorial\target\echo folder to this new folder.
3. If you have not already done so, from the Windows Start menu, choose Programs→Texas Instruments→Code Composer Studio 3.1→Code Composer Studio 3.1. (Or, double-click the Code Composer Studio icon on your desktop.)
4. Choose the Project→Open menu item. Open the echo.pjt project in the C:\CCStudio\_v3.10\myprojects\echo folder. (If another project was already open in Code Composer Studio, right-click on that project's .pjt file in the Project View and choose Close from the shortcut menu.)
5. Expand the Project View by clicking the + signs next to Projects, echo.pjt, DSP/BIOS Configuration, Generated Files, and Source.
6. Choose Project→Scan All File Dependencies.
7. The main source files used in this project are:
  - echo.c. This is the source code for the program.
  - echo.h. This is a header file included by echo.c to define various constants and structures. Click the + sign next to Include in the Project View to see a list that contains this file and DSP/BIOS module header files used by this program.
  - echo\_asm.s62. This assembly language source file contains the ECHO\_load macro, which simulates a load on the DSP by executing no-op instructions.

### Reviewing the Source Code

DSP/BIOS - PT2

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007

In Code Composer Studio's Project View area, double-click the echo.c, echo.h, and echo\_asm.s62 files in the Project View to open the source code files. Notice the parts of the example described in the following list.

You can click the file names below to see the section of code described.

Remember that this example focuses on scheduling issues, so the algorithms that process data do not do real work; they simply consume CPU cycles to simulate a processing load. In addition, no actual I/O is done and the code uses period() functions to simulate hardware interrupts.

**echo.h** . This header file defines constants used in the program.

**echo.c declarations** . The program includes a header file for each DSP/BIOS module used within this program. It also includes the echoCfg.h and echo.h header files. The echoCfg.h file is generated when the configuration file is saved; it contains external declarations for DSP/BIOS objects created in the configuration file.

**main function** . When the main function exits, the DSP/BIOS idle loop is executed. Within this loop, DSP/BIOS waits for events to occur.

**cancelerAlg function** . This function gets pointers to the next frames of input audio and output audio. It runs the cancelerAlg function to simulate the processing load. In the next step, you create a PRD object to run this function.

**decoderAlg function** . This function gets pointers to the next frames of input data and output audio. It runs the decoderAlg function to simulate the processing load. In the next step, you create a PRD object to run this function.

**encoderAlg function** . This function gets pointers to the next frames of input audio and output data. It runs the encoderAlg function to simulate the processing load. In the next step, you create a PRD object to run this function.

**cancelerLoad function** . This function is run by the canceler function to simulate processing required to cancel echoes. The function sends a message to the trace log if you changed the cancelerLoad variable and runs the ECHO\_Load function to perform the instructions to simulate the target CPU load.

**decoderLoad function** . This function is run by the decoder function to simulate processing required to decode data. The function sends a message to the trace log if you changed the decoderLoad variable and runs the ECHO\_Load function to perform the instructions to simulate the target CPU load.

**encoderLoad function** . This function is run by the encoder function to simulate processing required to encode data. The function sends a message to the trace log if you changed the encoderLoad variable and runs the ECHO\_Load function to perform the instructions to simulate the target CPU load.

**ECHO\_Load function** . This function is written in assembly language and stored in the echo\_asm.s62 file. It simulates a load on the DSP by executing a number of no-op instructions equal to 1000 times the load variable passed to the macro plus 11 overhead instructions. You can change the cancelerLoad, decoderLoad, and encoderLoad variables at run-time when you test this program.



### Configuring Objects to Run the Threads

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007

### Getting Started With the Code Composer Studio Tutorial

In this step, you create three PRD objects to run the **canceler** , **encoder** , and **decoder** functions.

1. Within Code Composer Studio, choose File→New→DSP/BIOS Configuration. Select the **template** for your DSP target and click OK.
2. If you are using a simulator target and did not select a configuration template specifically for the simulator, **set the RTDX Mode to Simulator**.
3. Click the + sign next to the Instrumentation and Scheduling categories to display their lists of modules.
4. Right-click the CLK - Clock Manager object and choose Properties from the context menu to open its property dialog. This dialog controls the high resolution and low resolution clock interrupt rates.
5. Change the Microseconds/Int property to 2500 and press the tab key. Notice that the PRD Register (on-chip timer period register) and Instructions/Int automatically change to new values that depend upon the speed of your DSP chip. When you use the on-chip timer to control PRD object execution, the low-resolution clock interrupt rate is the shortest interval at which a PRD object can perform its function.
6. Click OK to save your changes.
7. **Create three PRD objects** with the following **names** and **properties** . The Resulting period will automatically set with the period (ticks) update. Other properties should retain their default settings.

Object Name	period (ticks)	function	Resulting period (ms)
cancelerPrd	1	_canceler	2.5 milliseconds (400 Hz)
encoderPrd	9	_encoder	22.5 milliseconds (44.4 Hz)
decoderPrd	9	_decoder	22.5 milliseconds (44.4 Hz)

8. **Create** a LOG object and name it trace.
9. Change the **buffer** **property** of the LOG system LOG object to 512.

10. Choose File→Save. Save the file as echo.cdb in your working directory (usually C:\CCStudio\_v3.10\myprojects\echo). You are asked if you want to replace the existing file. Click Yes.

Saving this file generates the following files: echoCfg.cmd, echoCfg.h, echoCfg.h62, echoCfg.s62, and echoCfg.c.c.



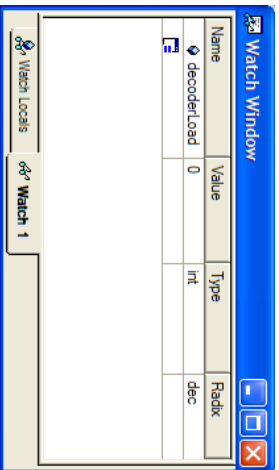
### Testing With CCSStudio

1. Choose Project→Build or click the  (Incremental Build) toolbar button.

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007

2. Choose File→Load Program. Select the echo out program you just built (usually in C:\CCStudio\_v3.10\myprojects\echo\Debug).
3. Choose Debug→Go Main.
4. Choose View→Watch Window.
5. Click on the Watch1 tab (instead of the Watch Locals tab).
6. Type decoderLoad in the first column.



7. Choose DSP/BIOS→RTA Control Panel. Verify that the checkboxes are all enabled.
8. Choose DSP/BIOS→Execution Graph.
9. Choose DSP/BIOS→CPU Load Graph. Note this feature will not function with simulators.
10. Choose DSP/BIOS→Statistics View.
11. Right-click on the Statistics View area and choose Property Page from the context menu. In the General tab, select cancelerPrd, encoderPrd, and decoderPrd. Click OK.
12. Choose DSP/BIOS→Message Log. Select trace in the Log Name list.
13. Arrange and resize the windows as needed to make them easy to see on your screen.
14. Choose File→Workspace→Save Workspace As. Save your window arrangement as echotest.wks in your working folder. You can reload this workspace file if you later need to restart Code Composer Studio.
15. Choose Debug→Run or press F5. Notice that the CPU level is low at this point.

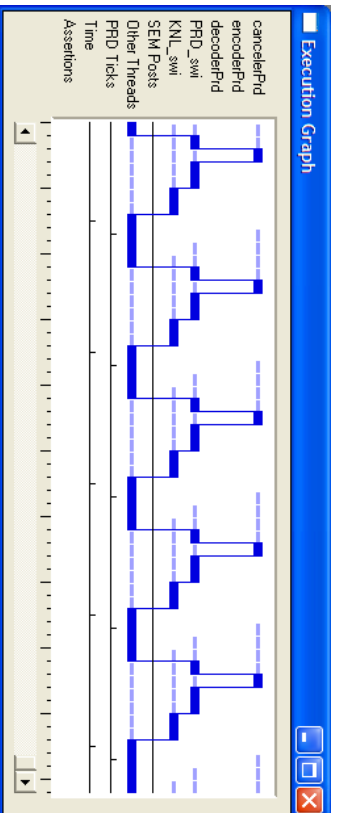
file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007

## Getting Started With the Code Composer Studio Tutorial

Page 384 sur 548

16. The Execution Graph and the Statistics View show that cancelerPrd is performed 9 times for every time the encoderPrd and decoderPrd are performed. In the Statistics View, the Count field increases because all three PRD objects are performing their functions. However, the Total, Max, and Average fields show zero. Why is this?



For PRD objects, the Total, Max, and Average fields report statistics in units of PRD ticks (not in instruction cycles as they do for SWI objects). Since the rate you set using Code Composer Studio was fairly long (2.5 milliseconds) and you have not yet increased the load on the encoder, decoder, or canceler functions, all these functions finish in less than one tick.

**Note:** The following load value is for a TMS320C6000 running at 100 MIPS. If your DSP is running at a different speed, multiply the load value given here by MIPS / 100. If you do not know what MIPS you are running at, open volumecdb and look at the Global Settings property called DSP Speed in MHz (CLKOUT). The Global Settings manager is in the System Settings category.

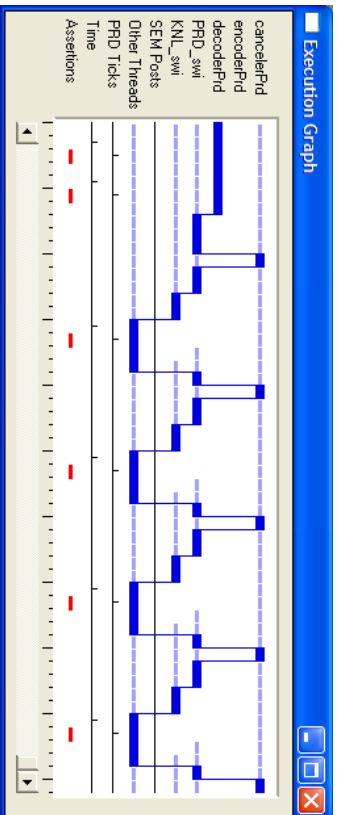
17. In the Watch Window, type 1000 in the Value column next to decoderLoad and click on the white space in the watch window to save the change. The value should immediately appear. Changing this variable causes the Echo\_Load function to use 1,000,000 instruction cycles when it is called by decoderAlg.

Notice that although the CPU load is not close to 100%, the Max field in the STS Data window shows that many ticks now pass before the cancelerPrd object can complete its function. This means the cancelerPrd no longer meets its real-time deadline, which is to complete within one tick. Although it is supposed to run every 1 tick (or 2.5 milliseconds), it is waiting for the decoder function to complete. The red squares in the assertions row of the Execution Graph show that DSP/BIOS detected that a thread missed its real-time deadline.

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007





18. In the Watch Window, change the decoderLoad to 10.

19. Then, right-click on the Statistics View and choose Clear from the shortcut menu. Notice that the Max value for the cancelerPrd is still high.

The Max value shown reflects the accumulation of missed deadlines for the PRD object. While most statistical information can be cleared, once a periodic function has missed a real-time deadline, the Max value returns to its high point as soon as it is recomputed. This is because the information used to compute the Max value still reflects the fact that the PRD object has missed deadlines. If the Max value becomes greater than the PRD object's period, you can divide the Max value by the period to determine how many real-time deadlines your PRD object has missed.

20. When you have finished experimenting with the decoderLoad, halt the program (Debug→Halt).

[What is the problem?](#)

[How can you solve the problem?](#)



### Correcting Scheduling Problems, Part 1

DSP/BIOS - PT2


Since the canceler operation is the most time-critical, we can try causing it to be executed by a SWI object instead of a PRD object.

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007

### Getting Started With the Code Composer Studio Tutorial

Page 386 sur 548

1. Open the echo.cdb file.
2. Click the + sign next to the Scheduling category to display its list of modules.
3. Open the **Properties** dialog for the cancelerPrd PRD object. Change the function property to `_SWI_post` and the arg0 property to `cancelerSw1`. This causes the cancelerPrd object to use the SWI\_post operation to post the cancelerSw1 object you create in the next step.
4. Add an SWI object called cancelerSw1.
5. Change the function **property** of this new object to `_canceler`.
6. Click once on the SWI - Software Interrupt Manager. Notice the list of SWI priorities and objects in the right half of the window. If you have not yet changed priority levels, KNL\_sw1 (which runs task threads) has a priority of 0 and all other SWI objects have a priority of 1.
7. Drag PRD\_sw1 to the Priority 2 folder. This allows periodic functions to interrupt the cancelerSw1.
8. Save your changes in the Configuration Tool (as echo.cdb).
9. Choose Project→Build or click the  (Incremental Build) toolbar button.
10. Choose File→Reload Program to reload the echo.out program you rebuilt.
11. Run the program again (Debug→Run). Make sure that the checkboxes in the RTA Control Panel are still enabled.
12. In the Watch Window, change the decoderLoad to 1000 (or 1000 \* MIPS / 100), and click on the white space in the watch window to save the change. The value should immediately appear. Notice in the Execution Graph and the Statistics View that the canceler thread still misses its real-time deadline.
13. Halt the program (Debug→Halt).



### Correcting Scheduling Problems, Part 2

DSP/BIOS - PT2

*Why did this not work?*

The PRD module manages periodic functions by following these steps:

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007

The PRD\_swi SWI is posted once for a PRD tick.\*  
 PRD\_swi loops through PRD objects:  
 If it is time for a PRD object to run:  
 Run that PRD object's function.  
 PRD\_swi continues loop of all PRD objects.

\*Note: PRD\_swi actually runs only if the number of ticks elapsed is equal to the greatest power of two among the common denominators of the PRD function periods. For example, if the periods of three PRD objects are 12, 24, and 36, PRD\_swi runs every 4 ticks. It does not simply run every 12 or 6 ticks because those intervals are not powers of two.

So the decoderPrd object is still running the decoder function, which takes a long time to execute, when the cancelerPrd PRD object needs to post the cancelerSwi software interrupt in order for it to be able to meet its real-time deadline.

Instead of making the highest priority function a software interrupt, we need to make the lower-priority functions into software interrupts. These functions may take longer to execute, but they are less time-critical, so they need to be able to yield to the canceler function. The functions performed directly by an application's PRD objects should take a total of less than a single PRD tick to execute.

To make these changes, follow these steps:

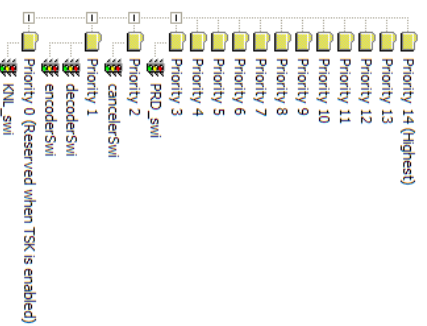
1. In the echo.cdb file, insert the following SWI objects, and set the function properties :

Object Name	function
encoderSwi	_encoder
decoderSwi	_decoder


2. Change the properties of the encoderPrd and decoderPrd PRD objects to the following values:

Object Name	period (ticks)	function	arg0
encoderPrd	9	_SWI_post	encoderswi
decoderPrd	9	_SWI_post	decoderswi

3. Highlight the SWI \_ Software Interrupt Manager. The right half of the echo.cdb window shows the priorities of the software interrupts. Use your mouse to drag the SWI objects so that they have the following priorities.



Note: When you create new priority levels, you may see a message that says "System stack size (See MEM) is too small to support a new SWI priority level." If you see this message, increase the value of the Stack Size property of the MEM - Memory Section Manager. Then you can add more priority levels. The Estimated Minimum Stack Size required by the program configuration is shown at the top of the echo.cdb window.

4. Save your changes in the Configuration Tool as echo.cdb.
5. Choose Project→Build or click the  (Incremental Build) toolbar button.
6. Choose File→Reload Program to reload the echo.out program you rebuilt.
7. Make sure that the checkboxes in the RTA Control Panel are still enabled.
8. Run the program again (Debug→Run).
9. In the Watch Window, change the decoderLoad to 1000 (or 1000 \* MIPS / 100), and click on the white space in the watch window to save the change. The value should immediately appear. Notice in the Execution Graph and the Statistics View that the canceler thread now meets its real-time deadline. You can tell because the Count field in the STS Data window increases but the Max field remains 0.
10. Halt the program (Debug→Halt).

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007

Getting Started With the Code Composer Studio Tutorial

Page 388 sur 548

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007

**Things to Try****DSP/BIOS - PT2**

- Open an STS Data window for the cancelerSwi, encoderSwi, and decoderSwi objects. Use the Watch Window to set the decoderLoad, encoderLoad, and cancelerLoad variables all to 15. Notice that the number of instructions in the Max field for the cancelerSwi is close to the cancelerLoad (multiplied by 1000), while the number of instructions to perform encoderSwi and decoderSwi is much higher than that. Why is this? Look at the Execution Graph to understand why. [Answer](#)
- Why does the Estimated Minimum Stack Size shown in the top of the echo.cdb window change when you add or remove SWI priority levels? [Answer](#)
- Use the Watch Window to increase the encoderLoad or cancelerLoad. How high can you set them before they miss their deadlines? [Answer](#)
- You can optimize the program by removing the cancelerSwi and making cancelerPrd execute the \_canceler function. This causes \_canceler to be executed at the same priority level as PRD\_sw1. It still has a higher priority than encoderSwi and decoderSwi. How much space on the system stack does this save? [Answer](#)

**See Also**

For a detailed description of thread types and thread interactions, read Chapter 4 of the DSP/BIOS User's Guide, which is provided in PDF format. For reference information, see the DSP/BIOS Modules topic in the online help.

This concludes the Scheduling SWI and PRD Threads lesson.

**Overview****DSP/BIOS - PT3**

This lesson shows how to cause task threads to yield to other tasks of equal priority levels.

A task is a type of thread. All tasks have priorities lower than hardware and software interrupts. Tasks differ from software interrupts in that a task can block its execution until some requirement is met that allows it to continue executing. In addition, tasks can yield the processor to other tasks with the same priority level.

Learning Objectives:

- Modify the project for DSP/BIOS

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007

**Getting Started With the Code Composer Studio Tutorial****Page 390 sur 548**

- Use DSP/BIOS module properties to prioritize task threads

- Use DSP/BIOS module properties to determine yielding and blocking of task threads

**Examples used in this lesson: tstkest**

Application Objective:

The example used in this lesson has three tasks of equal priority that use TSK\_yield for "round-robin" scheduling. TSK\_yield reschedules the current task behind other tasks of the same priority that are ready to run.

This lesson consists of the following steps. To go directly to a step, click on the link below:

- [Copying and Opening the Project](#)
- [Configuring Program Objects](#)
- [Reviewing the Source Code](#)
- [Predicting the Result](#)
- [Testing with CCStudio](#)
- [Things to Try](#)



The forward arrow will take you to the next page in this lesson.

**Copying and Opening the Project****DSP/BIOS - PT3**

1. Create a folder called tstkest in the C:\CCStudio\_v3.10\myprojects folder.
2. Copy all the files and folders from the C:\CCStudio\_v3.10\tutorial\target\tstkest folder to this new folder.
3. If you have not already done so, from the Windows Start menu, choose Programs→Texas Instruments→Code Composer Studio 3.1→Code Composer Studio 3.1. (Or, double-click the Code Composer Studio icon on your desktop.)
4. Choose the Project→Open menu item. Open the tstkest.prj project in the C:\CCStudio\_v3.10\myprojects\tstkest folder. (If another project was already open in Code Composer Studio, right-click on that project's .prj file in the Project View and choose Close from the shortcut menu.)

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007

5. Expand the Project View by clicking the + sign next to Projects, tskest.plt, DSP/BIOS Configuration, Generated Files, and Source.
6. The files used in this project are:
  - tskest.c. This is the source code for the program.
  - tskest.cdb. This is the configuration file you create in the next step. Saving this file generates the following files: tskestctrg.cmd, tskestctrg.h, tskestctrg.h62, tskestctrg.s62, and tskestctrg\_c.c.



### Configuring Program Objects

DSP/BIOS - PT3

In this step, you create three TSK objects to run the [task function](#) and a LOG object to which messages are printed.

1. Within Code Composer Studio, choose File→New→DSP/BIOS Configuration. Select the [template](#) for your DSP target and click OK.
2. If you are using a simulator target and did not select a configuration template specifically for the simulator, [set the RTDX Mode to Simulator](#).
3. Click the + sign next to the Instrumentation and Scheduling categories to display their lists of modules.
4. [Create three TSK objects with the following names and properties](#) . Other properties should retain their default settings. Task functions and arguments items are found under the Function tab of the Property Page.

Object Name	Task function	Task function argument
task0	_task	0
task1	_task	1
task2	_task	2

5. Highlight the TSK - Task Manager item in the left half of the window. Notice that the right half of the window shows that all three of the TSK objects you created have a priority level of one (1). The TSK\_idle object has a priority of zero (0), the lowest priority. This object causes the idle thread to run only when no other threads need to run.
6. [Create a LOG object](#) and name it trace.
7. Change the [bufferen property](#) of the trace LOG object to 512.
8. Change the [bufferen property](#) of the LOG\_system object to 512 and the logtype to fixed.

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007

### Getting Started With the Code Composer Studio Tutorial

Page 392 sur 548

9. Choose File→Save. Save the file as tskest.cdb in your working directory (usually C:\CCStudio\_v3.10\myprojects\tskest). You are asked if you want to replace the existing file. Click Yes.



### Reviewing the Source Code

DSP/BIOS - PT3

In Code Composer Studio's Project View area, double-click the tskest.c file in the Project View to see the source code. Notice the parts of the example described in the following list.

You can click the code names below to see the section of code described.

- [declarations](#) . The program includes the std.h DSP/BIOS header file and the log.h and tsk.h header files. This allows the program to use the LOG and TSK modules. It also includes the tskestctrg.h header file, which contains external declarations for DSP/BIOS objects created in the configuration file. It defines NLOOPS as a constant for the number of times to run the loop in the task function.
- [main function](#) . When the main function exits, the DSP/BIOS idle loop is executed. Within this loop, DSP/BIOS waits for events to occur.

- [task function](#) . This function runs a for loop that prints a message to the log and then yields to the other tasks of the same priority by calling TSK\_yield. After looping NLOOPS times, the function prints a message and ends. In the previous step, you specified that this function is to be run by all three TSK objects, which pass a value from 0 to 2 to the function.

Notice that the argument to this function has a data type of Arg. This DSP/BIOS data type is capable of holding both Ptr and Int arguments. If your code needs to run on or be portable to the TMS320C55x DSP platform, you should use the Arg data type for arguments to any functions specified in the DSP/BIOS Configuration Tool. This includes functions called by SWI, TSK, HST, PIP, and PRD objects. The ArgToInt function converts the Arg data type to an integer.



### Predicting the Result

DSP/BIOS - PT3

In this example, three separate tasks with the same priority level run the task function.

What do you expect the trace log to contain when you run the program? Click the arrow below the answers to move to the next step.

Answer A

Answer B

Answer C

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007

Loop 0: Task 0 Working	Loop 0: Task 0 Working	Loop 0: Task 0 Working
Loop 1: Task 0 Working	Loop 0: Task 1 Working	Loop 1: Task 1 Working
Loop 2: Task 0 Working	Loop 3: Task 0 Working	Loop 2: Task 2 Working
Loop 3: Task 0 Working	Loop 0: Task 2 Working	Loop 3: Task 0 Working
Loop 4: Task 0 Working	Task 0 DONE	Loop 4: Task 1 Working
Task 0 DONE	Loop 1: Task 0 Working	Task 0 DONE
Loop 0: Task 1 Working	Loop 1: Task 1 Working	Task 1 DONE
Loop 1: Task 1 Working	Loop 2: Task 0 Working	Task 1 DONE
Loop 2: Task 1 Working	Loop 2: Task 1 Working	Task 2 DONE
Loop 3: Task 1 Working	Loop 2: Task 2 Working	
Loop 4: Task 1 Working	Loop 3: Task 2 Working	
Task 1 DONE	Loop 3: Task 0 Working	
Loop 0: Task 2 Working	Loop 3: Task 1 Working	
Loop 1: Task 2 Working	Loop 3: Task 2 Working	
Loop 2: Task 2 Working	Loop 4: Task 1 Working	
Loop 3: Task 2 Working	Loop 4: Task 2 Working	
Loop 4: Task 2 Working	Task 0 DONE	
Task 2 DONE	Task 1 DONE	
	Task 2 DONE	

Answer A: The program runs the entire task function for the first task, then the entire task function for the second task, then the entire task function for the third task.

Answer B: The program prints all three messages for Loop 0, then all three messages for Loop 1, and so on.

Answer C: The program prints a message within Loop 0 for Task 0, within Loop 1 for Task 1, and so on.

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007

Getting Started With the Code Composer Studio Tutorial


Page 394 sur 548



### Testing With CCSStudio

DSP/BIOS - PT3

Run the program to see if your expectations were correct.

1. Choose Project→Build or click the  (Incremental Build) toolbar button.
2. Choose File→Load Program. Select the tskest.out program you just built (usually in C:\CCStudio\_v3.10\myprojects\tskest\Debug).
3. Choose DSP/BIOS→Message Log.
4. Select trace in the Log Name list.
5. Choose DSP/BIOS→Execution Graph.
6. Choose DSP/BIOS→RTA Control Panel. Verify that there are check marks in the boxes to enable SWI, PRD, CLK, and TSK logging. Also, make sure the Global host enable box is checked.
7. Arrange and resize the windows as needed to make them easy to see on your screen.
8. Choose Debug→Run.
9. Compare the messages in the Message Log to your expectations.

Each task yields to the other tasks after printing one message. So the correct answer in the previous step was Answer B.

**Note:** Within a loop, tasks may not run in numeric order (task0, task1, task2). The order is determined by the order in which you created them in the Configuration Tool.



Things to Try

DSP/BIOS - PT3

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007

- You may have noticed that updates to the Execution Graph stopped soon after the tasks ran. Why is this? [Answer](#)
- Choose DSP/BIOS--Kernel/Object View. Move to the TSK tab. What information does this tab provide? [Answer](#)
- Open the configuration file and increase the priority of one of the tasks. Save the configuration file and rebuild the program. How does this affect the order of the messages printed in the log? [Answer](#)
- What happens if you set the priority of one of the tasks to -1 in the configuration file? [Answer](#)
- Does a task that is preempted by a higher-priority thread go to the front or the end of the queue for its priority level? For example, suppose you assign a higher priority to task 2. If task 0 is running and task 1 is ready to run when task 2 preempts task 0, which task runs after task 2 is finished? You can use the TSK\_setprt function to change the priority of a task programmatically. [Answer](#)

#### See Also

To learn more about task threads, read Chapter 4 of the DSP/BIOS User's Guide, which is provided in PDF format. For reference information, see the DSP/BIOS Modules topic in the online help.

This concludes the Using Tasks for Blocking and Yielding lesson.

## Introduction

DSP/BIOS - RS

Often the threads in a multi-threaded program need to coordinate access to shared resources, such as data and CPU processing. DSP/BIOS provides a number of structures that can be used for these purposes. The structures used depend on whether the threads need to share data or synchronize their activities:

- Share Data: If multiple threads need access to a global variable or object, they must coordinate access to such shared data. Otherwise, there may be confusion between the value currently stored in that location and the value a thread expects it is working with.
- Synchronize Activation: If threads need to run when various resources are available, they must send and receive messages in response to resource availability. For example, a thread may need to run only when two buffers of data have been filled with new data.

The structures you can use to coordinate data access and activation depend upon the types of threads you are using as shown in the following table:

Thread	Share Data Via:	Synchronize Activation Via:
HWT	streams, queues, pipes, global variables	not applicable
SWI	streams, queues, pipes, global variables	SWI mailbox
TSK	streams, queues, pipes, locks, mailboxes, global variables	semaphores, mailboxes

file:///C:/Documents and Settings/Pierre-Armand/Local Settings/Temp/~hhBA53.htm

26/09/2007

## Getting Started With the Code Composer Studio Tutorial

Page 396 sur 548

### IDL

streams, queues, pipes, global variables

not applicable

This section contains the following lessons on synchronizing and sharing program resources:

[Using Semaphores to Send Messages](#)

[Using Semaphores for Mutual Exclusion](#)

[Using Mailboxes to Send Messages](#)

#### See Also

For additional information about resource sharing, see the following:

- Chapter 4 of the DSP/BIOS User's Guide, which is provided in PDF format
- The DSP/BIOS Modules topic in the online help
- Various DSP/BIOS application notes available on the Texas Instruments website, including:
  - Understanding the Functional Enhancements of DSP/BIOS and their Utilization in Real-Time DSP Applications (SPRA648)

## Overview

DSP/BIOS - RS1

This lesson shows how to use semaphores to synchronize access to a message queue. A semaphore is a data structure used for inter-task communication and to synchronize thread execution and access to shared data structures. When a task contends on a semaphore, that task waits for another thread to post the semaphore. While the task is waiting, other threads can run.

Learning Objectives:

- Modify the project for DSP/BIOS
- Use semaphores to synchronize threads
- Use semaphores to prioritize tasks

file:///C:/Documents and Settings/Pierre-Armand/Local Settings/Temp/~hhBA53.htm

26/09/2007

**Examples used in this lesson: semtest**

Application Objective:

The example used in this lesson uses a semaphore to allow three tasks to take turns writing to a queue.

This lesson consists of the following steps. To go directly to a step, click on the link below:

[About Semaphores](#)

[Copying and Opening the Project](#)

[Configuring Program Objects](#)

[Reviewing the Source Code](#)

[Predicting the Result](#)

[Testing with CCSStudio](#)

[Things to Try](#)



The forward arrow will take you to the next page in this lesson.

**About Semaphores****DSP / BIOS - RS1**

A semaphore is a data structure used for inter-task communication and to synchronize thread execution and access to shared data structures. Semaphores are often used to coordinate access to a shared resource among a set of competing tasks. They can also be used to exclude mutual execution of protected code sections.

SEM objects in DSP/BIOS are counting semaphores. Counting semaphores keep an internal count of the number of corresponding resources available.

To post a semaphore, a task calls SEM\_post. This increments the count of the semaphore.

To pend on a semaphore, a task calls SEM\_pend. If the semaphore count is greater than 0, SEM\_pend simply decrements the count and returns. If the semaphore count is 0, that task waits for another thread to call SEM\_post to post the semaphore. While the task is waiting, other threads can run. The timeout parameter to SEM\_pend allows the task to wait until a timeout, to wait indefinitely (SYS\_FOREVER), or to not wait at all (0).

In general, the semaphore count keeps track of the number of resources available. At the beginning of the semtest program, no resources are available because no writer task has put

[file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm](#)

26/09/2007

**Getting Started With the Code Composer Studio Tutorial**

Page 398 sur 548

messages on the msgQueue. So, the initial semaphore count is zero. The reader task first pends on the semaphore. Since the initial count is 0, it blocks and allows a writer task to run. A writer task put a message on the msgQueue and posts the semaphore. This allows the reader task to resume running and to get the message from msgQueue.

If multiple tasks are pending on the same semaphore, the first task to pend on the semaphore is given access to the semaphore when another thread posts the semaphore. This task is not necessarily the highest-priority task.

**Copying and Opening the Project****DSP / BIOS - RS1**

Begin by following these steps to copy the project to a working folder and open the project with Code Composer Studio:

1. Create a folder called semtest in the [C:\CCStudio\\_v3.10\myprojects](#) folder.
2. Copy all the files and folders from the [C:\CCStudio\\_v3.10\tutorial\target\semtest](#) folder to this new folder.
3. If you have not already done so, from the Windows Start menu, choose Programs → Texas Instruments → Code Composer Studio 3.1 → Code Composer Studio 3.1. (Or, double-click the Code Composer Studio icon on your desktop.)
4. Choose the Project → Open menu item. Open the semtest.pjt project in the [C:\CCStudio\\_v3.10\myprojects\semtest](#) folder. (If another project was already open in Code Composer Studio, right-click on that project's .pjt file in the Project View and choose Close from the shortcut menu.)
5. Expand the Project View by clicking the + signs next to Projects, semtest.pjt, DSP/BIOS Configuration, Generated Files, and Source.

The files used in this project are:

- semtest.c. This is the source code for the program.
- semtest.cdb. This is the configuration file you create in the next step. Saving this file generates the following files: semtestcfg.cmd, semtestcfg.h, semtestcfg.hex2, semtestcfg.s62, and semtestcfg\_c.c

**Configuring Program Objects**

[file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm](#)

26/09/2007

In this step, you create a SEM object, five TSK objects, and a LOG object to which messages are printed. The two QUE objects used in this example are created dynamically by the semtestc program.

1. Within Code Composer Studio, choose File->New->DSP/BIOS Configuration. Select the **template** for your DSP target and click OK.
2. If you are using a simulator target and did not select a configuration template specifically for the simulator, **set the RTDX Mode to Simulator**.
3. Click the + sign next to the Instrumentation, Scheduling, and Synchronization categories to display their lists of modules.
4. **Create five TSK objects** with the following **names** and **properties**. Object name and Priority are under the General tab. Task functions and arguments are under the Function tab. Other properties should retain their default settings.

Object Name	Priority	Task function	Task function argument
initTsk	15	_initTsk	0
reader0	2	_reader	0
writer0	1	_writer	0
writer1	1	_writer	1
writer2	1	_writer	2

5. Highlight the TSK - Task Manager item in the left half of the window. Notice that the right half of the window shows the priority levels for the TSK objects you created. The TSK\_idle object has a priority of zero (0), the lowest priority. This object causes the idle thread to run only when no other threads need to run.
6. **Create** a SEM object and name it sem. The default Initial semaphore count value of zero (0) is correct.
7. **Create** a LOG object and name it trace. Change the **bufLen property** of the trace LOG object to 256.
8. Change the **logType property** of the LOG\_system LOG object to fixed and the **bufLen property** to 512.
9. Choose File->Save. Save the file as semtest.cdb in your working directory (usually C:\CCStudio\_v3.10\myprojects\seatest). You are asked if you want to replace the existing file. Click Yes.



### Reviewing the Source Code

In Code Composer Studio's Project View area, double-click the semtest.c file in the Project View to see the source code. Notice the parts of the example described in the following list.

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007

### Getting Started With the Code Composer Studio Tutorial

You can click the file names below to see the section of code described.

**declarations**. The declarations section of the program does the following:

- Includes the std.h header file and header files for DSP/BIOS modules used in the program.
- Includes the semtestCfg.h header file, which is generated when the configuration file is saved. This file contains external declarations for DSP/BIOS objects created in the configuration file.
- Defines NUMMSGs as a constant for the number of message objects to allocate.
- Defines NUMWRITERS as the number of writer task objects you created in the configuration file.
- Defines a MsgObj structure, which contains a QUE element, an integer value to identify the writer task, and a character value.
- Creates two QUE objects dynamically by declaring variables of type QUE\_Obj.

**main function**. This function simply prints a message to the log. When the main function exits, the DSP/BIOS idle loop is executed. Within this loop, DSP/BIOS waits for events to occur.

**initTask function**. This function initializes the two QUE objects, enables tracing programmatically, allocates space for messages, and puts NUMMSGs (3) messages on the freeQueue queue. This task runs first since it has the highest priority.

**reader function**. This function loops NUMMSGs \* NUMWRITERS times (9). Within the loop, it calls SEM\_pend to wait for the semaphore to be posted by a writer task. The lower-priority writer tasks can run while the reader0 task is waiting. After a writer task posts the semaphore, the reader function resumes execution. It gets the message from the msgQueue, prints a message to the log, and puts the message back on the freeQueue.

**writer function**. This function gets a message from the freeQueue, fills in message values, and puts the message on the msgQueue. It then posts the semaphore to allow the reader function to resume execution. The message values are the id number passed to the function by the task object and a character value from a to c.



### Predicting the Result

In this example, three **writer** tasks put messages in a queue that is read by a single **reader** task. A semaphore is used to synchronize access to the queue.

What do you expect the trace log to contain when you run the program? Click the arrow below to move to the next step.

Answer A	Answer B	Answer C
semtest example started (0) writing 'a' ... read 'a' from (0). (0) writing 'b' ... read 'b' from (0).	semtest example started (0) writing 'a' ... read 'a' from (0). (1) writing 'a' ... read 'a' from (1).	semtest example started (0) writing 'a' ... read 'a' from (0). (0) writing 'b' ... read 'a' from (0).

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007



```

(0) writing 'c' ...
read 'c' from (0).
writer (0) done.
(1) writing 'a' ...
read 'a' from (1).
(1) writing 'b' ...
read 'b' from (1).
(1) writing 'c' ...
read 'c' from (1).
writer (1) done.
(2) writing 'a' ...
read 'a' from (2).
(2) writing 'b' ...
read 'b' from (2).
(2) writing 'c' ...
read 'c' from (2).
reader done.
writer (2) done.

```

```

(2) writing 'a' ...
read 'a' from (2).
(0) writing 'b' ...
read 'b' from (0).
(1) writing 'b' ...
read 'b' from (1).
(2) writing 'b' ...
read 'b' from (2).
(0) writing 'c' ...
read 'c' from (0).
writer (0) done.
(1) writing 'c' ...
read 'c' from (1).
writer (1) done.
(2) writing 'c' ...
read 'c' from (2).
reader done.
writer (2) done.

```

Answer A: The program first alternates between reader0 and writer0 until writer0 has written all three messages. Then, the same synchronization occurs, in turn, with writer1 and writer2.

Answer B: The program first alternates between reader0 and all the writer tasks. Each writer task writes one message before yielding to the next writer task.


Answer C: The a single writer task writes three messages, then the reader task reads those three messages. This sequence continues for all three writer tasks.



### Testing With CCSstudio

DSP/BIOS - RS1

Run the program to see if your expectations were correct.

1. Choose Project→Build or click the  (Incremental Build) toolbar button.
2. Choose File→Load Program. Select the semtest.out program you just built (usually in C:\CCStudio\_V3.10\myprojects\semtest\Debug).
3. Choose DSP/BIOS→Message Log. Select trace in the Log Name list.
4. Choose DSP/BIOS→Execution Graph.

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007

### Getting Started With the Code Composer Studio Tutorial

Page 402 sur 548

5. Choose DSP/BIOS→RTA Control Panel.
6. **Arrange and resize** the windows as needed to make them easy to see on your screen.
7. Choose Debug→Run. Notice that the program sets the trace options in the RTA Control Panel.
8. Compare the messages in the Message Log to your expectations. Scroll the Execution Graph to the left so that you can see the sequence in which the tasks were executed.

The semaphore synchronizes access to the queues. The writer tasks do not yield to other writer tasks with the same priority levels because TSK\_yield is not called. As a result, execution alternates between the reader task and one of the writer tasks. When the first writer task has finished writing all of its messages, the next writer task begins alternating with the reader task. So the correct answer in the previous step was Answer A.

**Note:** Tasks may not run in numeric order (writer0, writer1, writer2). The order is determined by the order in which you created them in the Configuration Tool.



### Things to Try

DSP/BIOS - RS1

- You may have noticed that updates to the Execution Graph stopped soon after the tasks ran. Why is this? [Answer](#)
- Choose DSP/BIOS→Kernel/Object View. Rerun the program and look at the SEM tab. What information does this tab provide? [Answer](#)
- Edit the semtest.c file and remove the comments around the call to TSK\_yield in the [writer function](#). Rebuild and run the program. How does this affect the order of the messages printed in the log? [Answer](#)
- Open the configuration file and experiment with changes to the task priorities. Save the configuration file and rebuild the program. How does this affect the order of the messages printed in the log? [Answer](#)
- In this example, you used a semaphore to protect data shared by several tasks. Can you use semaphores to protect data shared by a task and a hardware interrupt? What about a task and a software interrupt? In such situations, which thread must call SEM\_pend, and which must call SEM\_post? Does execution control return to the task thread if a software or hardware interrupt calls SEM\_post? [Answer](#)
- Does the QUE manager pass the MsgObj structure by reference or by value? Why or why not? [Answer](#)
- How would the behavior of this program change if the calls to SEM\_pend and SEM\_post were removed? [Answer](#)
- What would happen if this program used QUE\_enqueue instead of QUE\_put and QUE\_dequeue instead of QUE\_get? [Answer](#)
- How does the semtest.c code create the QUE objects? [Answer](#)

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007

**See Also**

To learn more about semaphores, read Chapter 4 of the DSP/BIOS User's Guide, which is provided in PDF format. For reference information, see the DSP/BIOS Modules topic in the online help.

This concludes the Using Semaphores to Send Messages lesson.

**Overview****DSP/BIOS - RS2**

This lesson shows how to use semaphores to prevent one thread from accessing a shared data structure while another thread is modifying that structure.

Learning Objectives:

- Modify the project for DSP/BIOS
- Use semaphores to mutually exclude tasks from running during critical processing
- Use DSP/BIOS tools to view the effects

**Examples used in this lesson: mutex**

Application Objective:

The example used in this lesson uses a two tasks and a semaphore. Both tasks have sections of code protected by calls to SEM\_pend using the same semaphore. During execution of this protected section of code in the first task, the other task cannot perform its protected section of code even if it preempts the first task.

This lesson consists of the following steps. To go directly to a step, click on the link below:

[About Semaphores](#)

[Copying and Opening the Project](#)

[Configuring Program Objects](#)

[Reviewing the Source Code](#)

[file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm](#)

26/09/2007

Getting Started With the Code Composer Studio Tutorial

Page 404 sur 548

[Predicting the Result](#)

[Testing with CCStudio](#)

[Things to Try](#)



The forward arrow will take you to the next page in this lesson.

**About Semaphores****DSP/BIOS - RS2**

A semaphore is a data structure used for inter-task communication and to synchronize thread execution and access to shared data structures. Semaphores are often used to coordinate access to a shared resource among a set of competing tasks. They can also be used to exclude mutual execution of protected code sections.

SEM objects in DSP/BIOS are counting semaphores. Counting semaphores keep an internal count of the number of corresponding resources available.

To post a semaphore, a task calls SEM\_post. This increments the count of the semaphore.

To pend on a semaphore, a task calls SEM\_pend. If the semaphore count is greater than 0, SEM\_pend simply decrements the count and returns. If the semaphore count is 0, that task waits for another thread to call SEM\_post to post the semaphore. While the task is waiting, other threads can run. The timeout parameter to SEM\_pend allows the task to wait until a timeout, to wait indefinitely (SYS\_FOREVER), or to not wait at all (0).

In general, the semaphore count keeps track of the number of resources available. At the beginning of the semtest program, no resources are available because no writer task has put messages on the msgQueue. So, the initial semaphore count is zero. The reader task first pends on the semaphore. Since the initial count is 0, it blocks and allows a writer task to run. A writer task put a message on the msgQueue and posts the semaphore. This allows the reader task to resume running and to get the message from msgQueue.

If multiple tasks are pending on the same semaphore, the first task to pend on the semaphore is given access to the semaphore when another thread posts the semaphore. This task is not necessarily the highest-priority task.

**Copying and Opening the Project****DSP/BIOS - RS2**

[file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm](#)

26/09/2007

Begin by following these steps to copy the project to a working folder and open the project with Code Composer Studio:

1. Create a folder called `mutex` in the [C:\CCStudio\\_v3.10\myprojects](#) folder.
2. Copy all the files and folders from the [C:\CCStudio\\_v3.10\tutorial\target\mutex](#) folder to this new folder.
3. If you have not already done so, from the Windows Start menu, choose Programs → Texas Instruments → Code Composer Studio 3.1 → Code Composer Studio 3.1. (Or, double-click the Code Composer Studio icon on your desktop.)
4. Choose the Project → Open menu item. Open the `mutex.pjt` project in the [C:\CCStudio\\_v3.10\myprojects\mutex](#) folder. (If another project was already open in Code Composer Studio, right-click on that project's .pjt file in the Project View and choose Close from the shortcut menu.)
5. Expand the Project View by clicking the + signs next to Projects, `mutex.pjt`, DSP/BIOS Configuration, Generated Files, and Source.

The files used in this project are:

- `mutex.c`: This is the source code for the program.
- `mutex.cdb`: This is the configuration file you create in the next step. Saving this file generates the following files: `mutexcfg.cmd`, `mutexcfg.h`, `mutexcfg.hex2`, `mutexcfg.s62`, and `mutexcfg_C.C`.



### Configuring Program Objects

DSP / BIOS - RS2

In this step, you create two TSK objects, a SEM object, and a LOG object to which messages are printed.

1. Within Code Composer Studio, choose File → New → DSP/BIOS Configuration. Select the [template](#) for your DSP target and click OK.
2. If you are using a simulator target and did not select a configuration template specifically for the simulator, [set the RTDX Mode to Simulator](#).
3. Click the + sign next to the Instrumentation, Scheduling, and Synchronization categories to display their lists of modules.
4. **Create two TSK objects** with the following **names** and **properties**: Object name and Priority are under the General tab. Task functions are under the Function tab. Other properties should retain their default settings.

Object Name	Priority	Task function
<code>mutexTsk1</code>	1	<code>_mutex1</code>

[file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm](#)

26/09/2007

### Getting Started With the Code Composer Studio Tutorial

Page 406 sur 548

- |                        |   |                      |
|------------------------|---|----------------------|
| <code>mutexTsk2</code> | 2 | <code>_mutex2</code> |
|------------------------|---|----------------------|
5. Highlight the TSK - Task Manager item in the left half of the window. Notice that the right half of the window shows the priority levels for the TSK objects you created. The TSK Idle object has a priority of zero (0), the lowest priority. This object causes the idle thread to run only when no other threads need to run.
  6. **Create a SEM object and name it sem.**
  7. In the [properties](#) dialog for the SEM object, set the Initial semaphore count value to 1 (one).
  8. **Create a LOG object and name it trace.**
  9. In the [properties](#) dialog for the trace LOG object, set the buflen value to 512.
  10. In the [properties](#) dialog for the LOG\_system object, set the buflen value to 512.
  11. Choose File → Save. Save the file as `mutex.cdb` in your working directory (usually [C:\CCStudio\\_v3.10\myprojects\mutex](#)). You are asked if you want to replace the existing file. Click Yes.



### Reviewing the Source Code

DSP / BIOS - RS2

In Code Composer Studio's Project View area, double-click the `mutex.c` file in the Project View to see the source code. Notice the parts of the example described in the following list.

You can click the file names below to see the section of code described.

[declarations](#): The declarations section of the program does the following:

1. Includes the `std.h` header file and header files for DSP/BIOS modules used in the program.
  2. Includes the `mutexcfg.h` header file, which is generated when the configuration file is saved. This file contains external declarations for DSP/BIOS objects created in the configuration file.
  3. Defines the functions in the program.
  4. Declares the maincount variable, which is used by both tasks and must be protected by mutual exclusion.
- [main function](#): This function simply exits, allowing the DSP/BIOS idle loop to execute. Within this loop, DSP/BIOS waits for events to occur.

[file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm](#)

26/09/2007

[mutex1 function](#) . Within an infinite loop, this function does the following:

1. Prints a message if the semaphore count is zero. While a check like this is not needed in a real program, it is a handy place to set a breakpoint for purposes of this example.
2. Pends on the semaphore. This prevents the mutex2 function from accessing the maincount variable during execution of the following protected block of code.
3. Sets a local variable equal to maincount.
4. Waits in a while loop for 2 system clock ticks. While this function is waiting, the mutex2 function wakes up and begin executing. When it pends on the semaphore, it blocks and waits for the semaphore to be posted.
5. Increments the tsk1count and tempvar variables, and sets maincount to tempvar.
6. Prints messages that show the tsk1count and maincount in hex form. Since LOG\_printf cannot handle values larger than an Int, this example breaks the LQUns values into two 16-bit values. It downshifts the value by 16 bits and casts the result as an Int to display the first four characters of a hex value. It does a bitwise AND of the value with the maximum Int value and casts the result as an Int to display the last four characters of a hex value. The %04x format strings cause the value to be displayed as a four-character hex value with leading zeros.
7. Posts the semaphore to permit access to the shared data.
8. Sleeps for 1 system clock tick.

[mutex2 function](#) . Within an infinite loop, this function does the following:

1. Prints a message if the semaphore count is zero. While a check like this is not needed in a real program, it is a handy place to set a breakpoint for purposes of this example.
2. Pends on the semaphore. This prevents the mutex1 function from accessing the maincount variable during execution of the following protected block of code.
3. Sets a local variable equal to maincount.
4. Increments the tsk2count and tempvar variables, and sets maincount to tempvar.
5. Prints messages that show the tsk2count and maincount in hex form as described for the mutex1 function.
6. Posts the semaphore to permit access to the shared data.
7. Sleeps for 1 system clock tick.



file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007

Getting Started With the Code Composer Studio Tutorial

Page 408 sur 548

### Predicting the Result

DSP/BIOS - RS2

In this example, the [mutex1](#) and [mutex2](#) functions both contain protected code sections. They use a semaphore to mutually exclude processing of these code sections.

What repeating pattern do you expect the trace log to contain when you run the program? Click the arrow below to move to the next step.

Answer A	Answer B
<pre>Running mutex2 function Sem blocked in mutex1 mutex1: loop 0x00000001; total count = 0x00000001 Running mutex1 function Sem blocked in mutex1 mutex2: loop 0x00000001; total count = 0x00000002 . . .</pre>	<pre>Running mutex2 function mutex2: loop 0x00000001; total count = 0x00000001 Running mutex1 function Running mutex2 function Sem blocked in mutex2 mutex1: loop 0x00000001; total count = 0x00000002 mutex2: loop 0x00000002; total count = 0x00000003 . . .</pre>

Answer A: Both the mutex1 and mutex2 functions block on the semaphore and both functions run once in the repeating pattern.


Answer B: Only the mutex2 function blocks on the semaphore and the mutex2 function runs twice for every time the mutex1 function runs.



### Testing With CCSstudio

DSP/BIOS - RS2

Run the program to see if your expectations were correct.

1. Choose Project→Build or click the  (Incremental Build) toolbar button.
2. Choose File→Load Program. Select the mutex.out program you just built (usually in C:\CCStudio\_v3.10\myprojects\mutex(Debug).
3. Choose DSP/BIOS→Message Log. Select trace in the Log Name list.

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007

4. Choose DSP/BIOS→RTA Control Panel. Verify that the Enable CLK logging, Enable TSK logging, and Global host enable boxes are checked. Remove the checkmark from the Enable SWI logging box.

Omitting the Enable SWI logging checkmark simplifies the Execution Graph in this example. This is because portions of the SEM\_post function are internally performed within the KNL\_swi thread. However, such switching between task threads and the KNL\_swi thread is not important to understanding mutual exclusion with semaphores.

5. Choose DSP/BIOS→Execution Graph.
6. Arrange and resize the windows as needed to make them easy to see on your screen.
7. Choose Debug→Run.
8. Compare the messages in the Message Log to your expectations. Look at the [Execution Graph](#) to examine the sequence in which the events occur.
9. To better understand the sequence of events, choose DSP/BIOS→Kernel/Object View. Set breakpoints in both mutex functions on the lines that call SEM\_post, SEM\_sleep, and the lines that set tempvar = maincount. Arrange the windows on your screen so that you can see the mutex.c file, Trace log, and Kernel/Object View windows. Rerun the program. At each breakpoint, look at the TSK and SEM tabs and the Trace log.

Because the mutex1 function waits in a while loop for two system clock ticks and the mutex2 function wakes up after one system clock tick, the mutex2 function blocks on the semaphore while waiting for the mutex1 function to post the semaphore. The reverse never happens since the protected processing block in the mutex2 function takes much less than one system clock tick to run. Also, because the mutex1 function waits for two system clock ticks, the mutex2 function runs twice as often as the mutex1 function. So the correct answer in the previous step was Answer B.



### Things to Try

DSP/BIOS - RS2

- In this example, you used a semaphore to mutually exclude tasks from running during critical processing. Can you use semaphores to mutually exclude a task thread and a hardware interrupt thread? What about a task and a software interrupt? [Answer](#)
- Comment out all four calls to SEM\_post and SEM\_sleep. How does the behavior of the program change? Look at the values displayed in the log for the loop counts and the total count. [Answer](#)
- Un-comment the calls to SEM\_post and SEM\_sleep, and comment out the two calls to TSK\_sleep. How does the behavior of the program change? Run the program for a few seconds. Then halt the program to view the log information. [Answer](#)
- Open the configuration file and experiment with changes to the task priorities. Save the configuration file and rebuild the program. How does this affect the order of the messages printed in the log? [Answer](#)

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007

### Getting Started With the Code Composer Studio Tutorial

Page 410 sur 548

- Compare the initial semaphore count used in the mutex example (1) to the initial semaphore count used in the [semtest example](#) (0). Why do these examples use different semaphore counts? [Answer](#)
- How long does the mutex1 function actually wait in the following while loop? By how much can this length of time vary? [Answer](#)

```
time = CLK_getTime();
while (CLK_getTime() <= (time + 1)) {
    ;
}
```

#### See Also

To learn more about semaphores, read Chapter 4 of the DSP/BIOS User's Guide, which is provided in PDF format. For reference information, see the DSP/BIOS Modules topic in the online help.

This concludes the Using Semaphores for Mutual Exclusion lesson.



### Overview

DSP/BIOS - RS3

This lesson shows how to use mailboxes to send messages between tasks. A mailbox is a data structure used to pass messages from one task to another. A fixed length shared mailbox can be used to ensure that the flow of incoming messages does not exceed the ability of the system to process those messages.

#### Learning Objectives:

- Modify the project for DSP/BIOS
- Use mailboxes to pass messages from one task to another
- Use DSP/BIOS tools to view the effects of mailbox messaging

#### Examples used in this lesson: mbxtest

Application Objective:

The example used in this lesson uses a mailbox that can contain up to three messages to coordinate the activities of a single reader task and three writer tasks.

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007

This lesson consists of the following steps. To go directly to a step, click on the link below:

- [Copying and Opening the Project](#)
- [Configuring Program Objects](#)
- [Reviewing the Source Code](#)
- [Predicting the Result](#)
- [Testing with CCSStudio](#)

Things to Try



The forward arrow will take you to the next page in this lesson.

### Copying and Opening the Project

DSP/BIOS - RS3

Begin by following these steps to copy the project to a working folder and open the project with Code Composer Studio:

1. Create a folder called `mbxtest` in the `C:\CCStudio_v3.10\myprojects` folder.
  2. Copy all the files and folders from the `C:\CCStudio_v3.10\tutorial\target\mbxtest` folder to this new folder.
  3. If you have not already done so, from the Windows Start menu, choose Programs→Texas Instruments→Code Composer Studio 3.1→Code Composer Studio 3.1. (Or, double-click the Code Composer Studio icon on your desktop.)
  4. Choose the Project→Open menu item. Open the `mbxtest.pjt` project in the `C:\CCStudio_v3.10\myprojects\mbxtest` folder. (If another project was already open in Code Composer Studio, right-click on that project's .pjt file in the Project View and choose Close from the shortcut menu.)
  5. Expand the Project View by clicking the + signs next to Projects, `mbxtest.pjt`, DSP/BIOS Configuration, Generated Files, and Source.
- The files used in this project are:
- `mbxtest.c`: This is the source code for the program.

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007

### Getting Started With the Code Composer Studio Tutorial

Page 412 sur 548

- `mbxtest.cdb`: This is the configuration file you create in the next step. Saving this file generates the following files: `mbxtestcfg.cmd`, `mbxtestcfg.h`, `mbxtestcfg.h62`, `mbxtestcfg.s62`, and `mbxtestcfg.c.c`.



### Configuring Program Objects

DSP/BIOS - RS3

In this step, you create an MBX object, four TSK objects, and a LOG object.

1. Within Code Composer Studio, choose File→New→DSP/BIOS Configuration. Select the **template** for your DSP target and click OK.
2. If you are using a simulator target and did not select a configuration template specifically for the simulator, **set the RTDX Mode to Simulator**.
3. Click the + sign next to the Instrumentation, Scheduling, and Synchronization categories to display their lists of modules.
4. **Create four TSK objects with the following names and properties**. Object name and Priority are under the General tab. Task functions are under the Function tab. Other properties should retain their default settings.

Object Name	Priority	Task Function	Task function argument
reader0	1	_reader	0
writer0	1	_writer	0
writer1	1	_writer	1
writer2	1	_writer	2

5. Highlight the TSK - Task Manager item in the left half of the window. Notice that the right half of the window shows the priority levels for the TSK objects you created. The TSK\_idle object has a priority of zero (0), the lowest priority. This object causes the idle thread to run only when no other threads need to run.

6. **Create an MBX object and name it mbx**. Change the Message Size **property** of this object to 5. Change the Mailbox Length property to 2. The `MsgObj` structure is declared as follows:

```
typedef struct MsgObj {
    Int id; /* writer task id */
    Char val; /* message value */
} MsgObj, *MsgF;
```

7. **Create a LOG object and name it trace**. Change the **bufLen property** of the trace LOG object to 256. On the C6000, this structure uses a total of 40 bits because an Int requires 32 bits and a Char requires 8 bits. The Message Size is measured in **MADUs**, which is 8 bits on the C6000. So the Message Size you use is 40 divided by 8, which is 5.

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007

8. Change the `logtype` property of the `LOG_system` LOG object to fixed and the `bufrlen` property to 512.
9. Choose File→Save. Save the file as `mbxtest.cdb` in your working directory (usually `C:\CCStudio_v3.10\myprojects\mbxtest`). You are asked if you want to replace the existing file. Click Yes.



## Reviewing the Source Code

DSP/BIOS - RS3

In Code Composer Studio's Project View area, double-click the `mbxtest.c` file in the Project View to see the source code. Notice the parts of the example described in the following list. You can click the file names below to see the section of code described.

**declarations** . The program includes the `std.h` DSP/BIOS header file and header files for the DSP/BIOS modules used in the program. It also includes the `mbxtestcfg.h` header file, which contains external declarations for DSP/BIOS objects created in the configuration file. It defines `NUMMSG` as a constant for the number of messages to allocate. It defines `TIMEDOUT` as the number of system clock ticks to wait for a message. The `MsgObj` structure contains an integer value to identify the writer task and a character value.

**main function** . When the main function exits, the DSP/BIOS idle loop is executed. Within this loop, DSP/BIOS waits for events to occur.

**reader function** . This function waits for a message to be available in the mailbox. While it is waiting, other threads can run. When it gets a message, it prints information from that message to the log. If it does not receive a message within `TIMEDOUT` (10) system clock ticks, it breaks out of the loop and completes its execution.

**writer function** . This function loops `NUMMSG` times. Within the loop, it writes values into a `MsgObj` structure, posts that message to the mailbox, and prints a message to the log. If the mailbox is full, it waits for space to be available. While it is waiting, other threads can run.



## Predicting the Result

DSP/BIOS - RS3

In this example, three **writer** tasks put messages in a mailbox that is read by a single **reader** task. The mailbox can contain a maximum of two messages.

What do you expect the trace log to contain when you run the program? Click the arrow below to move to the next step.



file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007

## Getting Started With the Code Composer Studio Tutorial

Page 414 sur 548

Answer A	Answer B	Answer C
<pre>(0) writing 'a' ... (0) writing 'b' ... (0) writing 'c' ... writer (0) done. read 'a' from (0). read 'b' from (0). read 'c' from (0). (1) writing 'a' ... (1) writing 'b' ... (1) writing 'c' ... writer (1) done. read 'a' from (1). read 'b' from (1). read 'c' from (1). (2) writing 'a' ... (2) writing 'b' ... (2) writing 'c' ... writer (2) done. read 'a' from (2). read 'b' from (2). read 'c' from (2). timeout expired for MBX_pend() reader done.</pre>	<pre>(0) writing 'a' ... read 'a' from (0). (0) writing 'b' ... read 'b' from (0). (0) writing 'c' ... writer (0) done. read 'c' from (0). (1) writing 'a' ... read 'a' from (1). (1) writing 'b' ... read 'b' from (1). (1) writing 'c' ... writer (1) done. read 'c' from (1). (2) writing 'a' ... read 'a' from (2). (2) writing 'b' ... read 'b' from (2). (2) writing 'c' ... writer (2) done. read 'c' from (2). timeout expired for MBX_pend() reader done.</pre>	<pre>(0) writing 'a' ... (0) writing 'b' ... read 'a' from (0). read 'b' from (0). (0) writing 'c' ... writer (0) done. (1) writing 'a' ... read 'c' from (0). read 'a' from (1). (2) writing 'a' ... read 'b' from (1). (2) writing 'b' ... read 'a' from (2). read 'c' from (1). (2) writing 'c' ... writer (2) done. read 'c' from (2). timeout expired for MBX_pend() reader done.</pre>

Answer A: The program writes three messages to the mailbox, then reads three messages, and so on.

Answer B: The program writes a single message to the mailbox, then reads the message, and so on.

Answer C: The program writes two messages to the mailbox, then reads two messages, and so on.



## Testing With CCSStudio

DSP/BIOS - RS3

Run the program to see if your expectations were correct.

1. Choose Project→Build or click the  (Incremental Build) toolbar button.

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007

2. Choose File→Load Program. Select the mbxtest.out program you just built (usually in C:\CCStudio\_v3.1.0\myprojects\mbxtest\Debug).
3. Choose DSP/BIOS→Message Log. Select trace in the Log Name list.
4. Choose DSP/BIOS→Execution Graph.
5. Choose DSP/BIOS→RTA Control Panel. Right-click on the RTA Control Panel area and choose Enable All from the context menu.
6. **Arrange and resize** the windows as needed to make them easy to see on your screen.
7. Choose Debug→Run.
8. Compare the messages in the Message Log to your expectations. Scroll the Execution Graph to the left so that you can see the sequence in which the tasks were executed.

The mailbox can contain two messages at once. When a writer task attempts to post a third message to the mailbox, the writer task must wait until the reader task removes the messages. When the reader task has removed both messages, it posts on the mailbox and waits for a writer task to post more messages. So the correct answer in the previous step was Answer C.

**Note:** Tasks may not run in numeric order (writer0, writer1, writer2). The order is determined by the order in which you created them in the Configuration Tool.



### Things to Try

- Choose DSP/BIOS→Kernel/Object View. Return the program and look at the MBX tab. What information does this tab provide? Try setting breakpoints at various points in the program to see the values in this tab change. [Answer](#)
- Edit the mbxtest.c file and remove the comments around the call to TSK\_yield in the [writer function](#) . Rebuild and run the program. How does this affect the order of the messages printed in the log? [Answer](#)
- Open the configuration file and experiment with changes to the task priorities. Save the configuration file and rebuild the program. How does this affect the order of the messages printed in the log? [Answer](#)

#### See Also

To learn more about mailboxes, read Chapter 4 of the DSP/BIOS User's Guide, which is provided in PDF format. For reference information, see the DSP/BIOS Modules topic in the online help.

This concludes the Using Mailboxes to Send Messages lesson.

file:///C:/Documents and Settings/Pierre-Armand/Local Settings/Temp/~hhBA53.htm

26/09/2007

## Getting Started With the Code Composer Studio Tutorial

Page 416 sur 548



### Introduction

#### DSP/BIOS - MEM

You can use the Configuration Tool to specify the memory segments use for the code and data sections of your application. The result is a logical memory map of the physical system. This layer of abstraction can simplify the process of migrating the application to another hardware platform or next-generation TMS320 device.

The MEM module also provides a set of run-time functions for allocating storage from one or more segments of memory.

This section contains the following lessons on managing memory in your DSP/BIOS program:

[Allocating and Freeing Memory](#)

#### See Also

For additional information about memory management, see the following:

- Chapter 5 of the DSP/BIOS User's Guide, which is provided in PDF format
- The DSP/BIOS Modules topic in the online help
- Various DSP/BIOS application notes available on the Texas Instruments website, including:
  - Programming and Debugging Tips for DSP/BIOS (SPRA640)
  - DSP/BIOS Sizing Guidelines on TMS320C6000/C5000 DSPs for Code Composer Studio2.0 (SPRA772)



### Overview

This lesson shows how to allocate and free memory from a heap that has been defined with the Configuration Tool.

#### DSP/BIOS - MEM

file:///C:/Documents and Settings/Pierre-Armand/Local Settings/Temp/~hhBA53.htm

26/09/2007



Learning Objectives:

- Modify the project for DSP/BIOS
- Use the DSP/BIOS Configuration Tool to allocate memory from a heap
- Use the DSP/BIOS Configuration Tool to free memory from a heap
- Use DSP/BIOS tools to view the effects

**Examples used in this lesson: memtest**

Application Objective:

The example used in this lesson prints statistics about memory usage before allocating, after allocating, and after freeing memory from a memory heap.

This lesson consists of the following steps. To go directly to a step, click on the link below:

[Copying and Opening the Project](#)

[Configuring Program Objects](#)

[Reviewing the Source Code](#)

[Predicting the Result](#)

[Testing with CCSStudio](#)

[Things to Try](#)



The forward arrow will take you to the next page in this lesson.

### Copying and Opening the Project

**DSP/BIOS - MEM**

Begin by following these steps to copy the project to a working folder and open the project with Code Composer Studio:

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007

### Getting Started With the Code Composer Studio Tutorial

Page 418 sur 548

1. Create a folder called memtest in the C:\CCStudio\_v3.10\myprojects folder.
2. Copy all the files and folders from the C:\CCStudio\_v3.10\tutorial\target\memtest folder to this new folder.
3. If you have not already done so, from the Windows Start menu, choose Programs →Texas Instruments→Code Composer Studio 3.1→Code Composer Studio 3.1. (Or, double-click the Code Composer Studio icon on your desktop.)
4. Choose the Project→Open menu item. Open the memtest.pjt project in the C:\CCStudio\_v3.10\myprojects\memtest folder. (If another project was already open in Code Composer Studio, right-click on that project's .pjt file in the Project View and choose Close from the shortcut menu.)
5. Expand the Project View by clicking the + signs next to Projects, memtest.pjt, DSP/BIOS Configuration, Generated Files, and Source.

The files used in this project are:

- memtest.c. This is the source code for the program.
- memtest.cdb. This is the configuration file you create in the next step. Saving this file generates the following files: memtestcfg.cmd, memtestcfg.h, memtestcfg.h62, memtestcfg.s62, and memtestcfg.c.c.



### Configuring Program Objects

**DSP/BIOS - MEM**

In this step, you configure a memory heap using a MEM object. You also create a TSK object and a LOG object.

1. Within Code Composer Studio, choose File→New→DSP/BIOS Configuration. Select the **template** for your DSP target and click OK.
2. If you are using a simulator target and did not select a configuration template specifically for the simulator, **set the RTDX Mode to Simulator**.
3. Click the + sign next to the System, Instrumentation, and Scheduling, categories to display their lists of modules.
4. Click on the MEM - Memory Section Manager to view the list of memory types for your target. Each target has its own specific regions of memory.
5. Open the **properties** dialog for the IDRAM MEM object. The memory sections listed may be different for your particular board.
6. Put a checkmark in the create a heap in this memory box. This creates a heap from which memory can be dynamically allocated by the application.
7. Change the heap size to 0x0400. This causes the size of the heap to be 1024 MADUs (Minimum Addressable Data Units). An MADU is the smallest unit of data storage that can be read

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007

or written by the CPU. For the c6000, this is an 8-bit byte.

8. Put a checkmark in the enter a user defined heap identifier label box.
9. Type `_SEG0` (the character after "SEFC" is a zero) as the heap identifier label. The `memset.c` program references the `SEG0` as an external declaration. The underscore prefix is required here because the Configuration Tool creates assembly source code that declares this identifier.
10. Create a **TSK object** with the name `initTsk` and the following **properties** . Set the Task function to `_initTask`. The Priority property should be 15. Other properties should retain their default settings. Object name and Priority are under the General tab. Task functions are under the Function tab. Notice that the `sysstack` size for this task is 1024 MADUs (on most targets).
11. Create a LOG object and name it trace. Change the buffer **property** of the trace LOG object to 128 words.
12. Change the buffer **property** of the LOG\_system LOG object to 256 words and the `logType` **property** to fixed.
13. Choose File→Save. Save the file as `memset.cdb` in your working directory (usually `C:\CCStudio_V3.10\myproject\memset`). You are asked if you want to replace the existing file. Click Yes.



## Reviewing the Source Code

### DSP/BIOS - MEM

In Code Composer Studio's Project View area, double-click the `memset.c` file in the Project View to see the source code. Notice the parts of the example described in the following list:

You can click the file names below to see the section of code described.

**declarations** . The program includes the `std.h` DSP/BIOS header file and header files for the DSP/BIOS modules used in the program. It also includes the `memsetcfg.h` header file, which contains external declarations for DSP/BIOS objects created in the configuration file. It defines the `NALLLOCS` constant as 2. This is the number of times to allocate memory from the heap. It defines the `BUFSIZE` constant as 128. This is the amount of memory to allocate from the heap during each call to `MEM_alloc` (in `MADUs`).

**main function** . This function simply prints a message to the log. When the main function exits, the DSP/BIOS idle loop is executed. Within this loop, DSP/BIOS waits for events to occur.

**initTask function** . This function uses the `MEM_alloc` function to allocate memory from the `SEG0` heap. It does this within a loop that executes `NALLLOCS` times. It then frees the same memory within a loop using the `MEM_free` function. Before and after each change to the allocated memory, this function prints a message to the log by either calling the `printmem` function or `LOG_print`. The calls to `LOG_print` display the numeric value of the `SEG0` identifier (which has no important meaning, but it happens to be 1) and the address of the block of memory that was allocated by `MEM_alloc`. This task runs first since it has the highest priority.

**printmem function** . This function gets the status of the `SEG0` memory segment by calling `MEM_stat`. This function returns `statbuf`, which has the following structure:

```
struct MEM_Stat {
    unsigned int  /* original size of segment */
    unsigned int  /* number of MADUs used in segment */
    unsigned int /* length of largest contiguous block */
}
```

This function then prints two messages using `LOG_print`. The first message shows the numeric value of the `SEG0` identifier and the size of the full memory heap. The second message shows the amount of memory that has been allocated in the heap and the largest block in the heap that is still available to be allocated. These size values are shown in `MADUs`.



## Predicting the Result

### DSP/BIOS - MEM

In this example, the [initTask function](#) allocates and frees memory. What is the result you expect to see when you run this program? Click the arrow below to move to the next step.

Answer A The program does not free all the memory it allocates.	Answer B The program allocates and then frees a total of 256 MADUs.	Answer C The program attempts to allocate more memory than is available.
--	--	---

Answer A: The program does not free all the memory it allocates.

Answer B: The program frees all the memory it allocates.


Answer C: The program attempts to allocate more memory than is available.



## Testing With CCSStudio

### DSP/BIOS - MEM

Run the program to see if your expectations were correct.

1. Choose Project→Build or click the  (Incremental Build) toolbar button.
2. Choose File→Load Program. Select the memtest.out program you just built (usually in C:\CCStudio\_v3.10\myprojects\memtest\Debug).
3. Choose DSP/BIOS→Message Log. Select Trace in the Log Name list.
4. Choose DSP/BIOS→Kernel/Object View.
5. Choose File→Open and open the memtest.c file.

6. Place breakpoints on the following lines of the `memtest.c` file by placing the editing cursor in that line and clicking the  (Toggle Breakpoint) toolbar button or pressing F9. This will set a software breakpoint. Your line numbers may vary:

- Line 50: `LOG_printf(trace, "allocating ...");`
- Line 69: `for (i = 0; i < NALLOCs; i++) {` (the beginning of the for loop that calls `MEM_free`)
- Line 72: `}` (the curly brace that ends the `initTask` function)

These lines are highlighted in blue in the `memtest.c` file.

7. **Arrange and resize** the windows as needed to make them easy to see on your screen.

8. Press Debug→Run or F5 to run to a breakpoint. At each breakpoint, look at the trace log, and click on the MEM item in the DSP/BIOS list in the Kernel/Object View. Scroll to the right in the Kernel/Object View to see the amount of Free Memory used. (In the Kernel/Object View, values that have changed since the last breakpoint are highlighted in red.) Continue pressing F5 until you have run through all the breakpoints.

- At the first breakpoint, 0x400 MADUs (1024 in decimal) are still free. (Depending on your target, the amount of free memory may be reduced by an overhead of up to 4 MADUs.)
- At the second breakpoint, only 0x300 MADUs (768 in decimal) are free. This value has changed by 256 MADUs, or `NALLOCs * BUFSIZE`. In addition to the amount of free and available space, the trace log shows the addresses at which the allocated memory begins. (Depending on your target, the amount of free memory may be reduced by an overhead of up to 4 MADUs.)
- At the last breakpoint, all 0x400 MADUs are available again. The trace message log window will say the test is complete. (Depending on your target, the amount of free memory may be reduced by an overhead of up to 4 MADUs.) So the correct answer in the previous step was Answer B.



### Things to Try

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007

### Getting Started With the Code Composer Studio Tutorial

Page 422 sur 548

#### DSP/BIOS - MEM

- In the Kernel/Object View, look at the other tabs. What information about memory usage do the other tabs provide? [Answer](#)
- Open the RTA Control Panel. Put checkmarks in all the logging boxes and the Global host enable box. Open the Execution Graph. Right-click on the Execution Graph and choose Clear from the context menu. Clear any breakpoints you have created. Reload and run the program. What do you notice about the interaction of the `KNL_Swi` and `initTask` threads? [Answer](#)
- Does the `MEM_alloc` function allocate memory beginning at the top or bottom of the memory heap? [Answer](#)
- What is the size of `memtest.out` program? How can you change the configuration file to reduce the size of the `memtest.out` program? [Answer](#)
- What would be the effect of calling `MEM_valloc` or `MEM_calloc` instead of `MEM_alloc`? How could you verify this effect? [Answer](#)
- Why is it better to call `MEM_alloc` than to call the standard `malloc` C function? [Answer](#)
- Open an MS-DOS Command window and run the `sectt.exe` utility with the following commands. What information does this utility show about the location of the memory you allocated? [Answer](#)

```
cd C:\CCStudio\myprojects\memtest
sectt.exe memtest.out
```

- Change the definition of the `NALLOCs` constant in `memtest.c` to 9. Then rebuild the program. What is the result when you run the program? Choose View→Memory and choose to look at the memory addresses allocated by the ninth call to `MEM_alloc`. [Answer](#)

#### See Also

To learn more about memory management, read Chapter 5 of the DSP/BIOS User's Guide, which is provided in PDF format. For reference information, see the DSP/BIOS Modules topic in the online help.

This concludes the Allocating and Freeing Memory lesson.



### Introduction

#### DSP/BIOS - IO

DSP/BIOS provides several modules that provide Input/Output support:

- The PIP and HST modules support the pipe I/O model, which generally supports low-level communication.

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007

- The SIO and DEV modules support the stream I/O model, which generally supports high-level, device-independent I/O.
- The RTDX module is used by DSP/BIOS to handle communication between the host PC and target DSP. You can use RTDX to implement additional host/target communication.

This section contains the following lessons on managing data I/O:

[Comparing I/O Models](#)

[Managing I/O with the Stream Model](#)

**See Also**

For additional information about handling I/O with DSP/BIOS, see the following:

- Chapter 7 and 8 of the DSP/BIOS User's Guide, which is provided in PDF format
- The DSP/BIOS Modules topic in the online help
- Various DSP/BIOS application notes available on the Texas Instruments website, including:
  - Understanding the Functional Enhancements of DSP/BIOS and their Utilization in Real-Time DSP Applications (SPRA648)
  - DSP/BIOS Technical Overview (SPRA646)
  - An Audio Example Using DSP/BIOS (SPRA598)
- DSP/BIOS by Degrees: Using DSP/BIOS in an Existing Application (SPRA591)



## Comparing I/O Models

**DSP/BIOS - IO**

For information about using the new IOM device driver model, see the DSP/BIOS Driver Developer's Guide (SPRU625).

DSP/BIOS supports two basic models for data transfer. The pipe model is used by the PIP and HST modules. The stream model is used by the SIO and DEV modules.

Both models require that a pipe or stream have a single reader thread and a single writer thread. Both models transfer buffers within the pipe or stream by copying pointers rather than by copying data between buffers. In general, the pipe model supports low-level communication, while the stream model supports high-level, device-independent I/O.

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007

## Getting Started With the Code Composer Studio Tutorial

Page 424 sur 548

The following table compares the two models in more detail.

Pipes (PIP and HST)	Streams (SIO and DEV)
<p>Programmer must create own driver structure.</p> <p>Reader and writer can be any thread type or host PC.</p> <p>PIP functions are non-blocking. Program must check to make sure a buffer is available before reading from or writing to the pipe.</p> <p>Uses less memory and is generally faster.</p> <p>Each pipe owns its own buffers.</p> <p>Pipes must be created statically with the Configuration Tool.</p> <p>No built-in support for stacking devices.</p> <p>Using the HST module with pipes is an easy way to handle data transfer between the host and target.</p>	<p>Provides a more structured approach to device-driver creation.</p> <p>One end must be handled by a task (TSK) using SIO calls. The other end is handled using Dxx calls. Typically the Dxx calls are performed within an HWI.</p> <p>SIO_put, SIO_get, and SIO_reclaim may be used as either blocking or non-blocking functions. When these functions block, a task waits until a buffer is available. (SIO_issue is non-blocking.)</p> <p>More flexible; generally simpler to use.</p> <p>Buffers can be transferred from one stream to another without copying. (In practice, copying is usually necessary anyway because the data is processed.)</p> <p>Streams can be created either at run time or statically with the Configuration Tool.</p> <p>Streams can be opened by name.</p> <p>Support is provided for stacking devices.</p> <p>A number of device drivers are provided with DSP/BIOS.</p>



## Overview

**DSP/BIOS - IO**

This lesson shows how to read data from a device using the SIO streaming model and print data to a log buffer.

Learning Objectives:

- Modify the project for DSP/BIOS
- Use the SIO module to support streaming I/O from the built-in DSP/BIOS device drivers
- Print data to the log buffer
- Use DSP/BIOS tools to view the effects

**Examples used in this lesson: sioatest1**

Application Objective:

The example used in this lesson is the sioatest1 example, which uses the built-in software generator driver (DGM) to generate sine wave data. The data is written to a DSP/BIOS log.

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007

This lesson consists of the following steps. To go directly to a step, click on the link below:

- [Copying and Opening the Project](#)
- [Configuring Program Objects](#)
- [Reviewing the Source Code](#)
- [Predicting the Result](#)
- [Testing with CCSStudio](#)
- [Things to Try](#)



The forward arrow will take you to the next page in this lesson.

### **Copying and Opening the Project**

---

#### **DSP/BIOS - IO**

Begin by following these steps to copy the project to a working folder and open the project with Code Composer Studio:

1. Create a folder called `sioest` in the `C:\CCStudio_V3.10\myprojects` folder.
2. Copy all the files and folders from the `C:\CCStudio_V3.10\tutorial\target\sioest` folder to this new folder.
3. If you have not already done so, from the Windows Start menu, choose Programs→Texas Instruments→Code Composer Studio 3.1→Code Composer Studio 3.1. (Or, double-click the Code Composer Studio icon on your desktop.)
4. Choose the Project→Open menu item. Open the `sioest1.pjt` project in the `C:\CCStudio_V3.10\myprojects\sioest` folder. (If another project was already open in Code Composer Studio, right-click on that project's .pjt file in the Project View and choose Close from the shortcut menu.)
5. Expand the Project View by clicking the + signs next to Projects, `sioest1.pjt`, DSP/BIOS Configuration, Generated Files, and Source.

**Note:** If you are using a C67X target, choose Project→Build Options. In the Compiler tab, select the Basic category. Set the Target Version to 670X. Using the correct target version is required in this lesson because the C67X DGN driver uses floating-point values to generate sine values.

The files used in this project are:

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007

### Getting Started With the Code Composer Studio Tutorial

Page 426 sur 548

- `sioest1.c`. This is the source code for the program.
- `sioest1.cdb`. This is the configuration file you create in the next step. Saving this file generates the following files: `sioest1cfg.cmd`, `sioest1cfg.h`, `sioest1cfg.h62`, `sioest1cfg.s62`, and `sioest1cfg_c.c`.



### **Configuring Program Objects**

---

#### **DSP/BIOS - IO**

In this step, you create and configure TSK, DGN, SIO, and LOG objects. You should be familiar with TSK and LOG objects from previous lessons.

The DGN driver is a built-in device driver provided with DSP/BIOS. It generates a signal using the properties you set. This signal can be a sine wave, random value, constant value, or a value provided by a user-defined function. This driver is used in this example because it does not rely on any external devices that are available on some, but not all, targets.

The SIO module supports streaming I/O from the built-in DSP/BIOS device drivers and from user-defined devices.

1. Within Code Composer Studio, choose File→New→DSP/BIOS Configuration. Select the **template** for your DSP target and click OK.
2. If you are using a simulator target and did not select a configuration template specifically for the simulator, set the **RTDX Mode to Simulator**.
3. Click the + sign next to the Instrumentation, Scheduling, and Input/Output categories to display their lists of modules.
4. In the Scheduling folder, create a **TSK object named streamTask**. Object name is under the General tab.
5. Set the following **properties** in the Function tab for the new streamTask object. Set the Task function to `_dosStreaming`. The Task function argument 0 property should be 3. Other properties should retain their default settings. When this task runs, it calls the `dosStreaming` function (the underscore is used because this is a C function) and passes 3 as the first argument.
6. In the Input/Output folder, right-click on the SIO - Stream Input and Output Manager and notice that you cannot choose Insert SIO at this time. This is because there are no device drivers available to bind to a stream.
7. In the Input/Output folder, click the + sign next to the Device Drivers.
8. For the DGN - Software Generator Driver, create a DGN object. Change its name to `sineWave`.
9. Open the **properties** dialog for this DGN object and select sine as the Device category. Selecting a particular Device category determines which properties you can set. Set the following additional properties for this sine generator. The default settings for other properties are correct.

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007

Property	Value
Gain	100
Frequency (Hz)	16

To improve performance, the DGN driver approximates the sine wave magnitude (maximum and minimum) value to the nearest power of two. This is done by computing a shift value by which each entry in the table is right-shifted before being copied into the input buffer. For example, setting the Gain to 100 causes the sine wave magnitude to be 128, the nearest power of two.

- Now that you have created a device driver, right-click on the SIO - Stream Input and Output Manager and choose Insert SIO. Change its name to InputStream.
- Set the following **properties** for the new InputStream object:

Property	Value
Device	sineWave
Mode	input
Number of buffers	2
Allocate Static Buffer(s)	yes (place a checkmark in this box)

DGN generator devices can be opened for input data streaming only; generators cannot be used as output devices.

- In the Instrumentation folder, create a LOG object and name it trace. Change the **buffer property** of the trace LOG object to 1024 words.
- Choose File→Save. Save the file as stotest1.cdb in your working directory (usually C:\CCStudio\_V3.1.0\myprojects\stotest). You are asked if you want to replace the existing file. Click Yes.



### Reviewing the Source Code

**DSP/BIOS - IO**

In Code Composer Studio's Project View area, double-click the stotest1.c file in the Project View to see the source code. Notice the parts of the example described in the following list.

You can click the file names below to see the section of code described.

**declarations** . The program includes the std.h DSP/BIOS header file and header files for the DSP/BIOS modules used in the program. It also includes the stotest1Cfg.h header file, which contains external declarations for DSP/BIOS objects created in the configuration file. It also declares a handle of type SIO\_Handle for use by DSP/BIOS functions that access the InputStream object.

**main function** . This function simply prints a message to the log. When the main function exits, the DSP/BIOS idle loop is executed. Within this loop, DSP/BIOS waits for events to occur.

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007

### Getting Started With the Code Composer Studio Tutorial

Page 428 sur 548

**doStreaming function** . This function first checks to see if there are any static buffers available in the stream. You caused DSP/BIOS to create these buffers by setting the Number of buffers property of the inputStream SIO object to 2. The function then runs the for loop 3 times because the first argument passed to the doStreaming function by the streamTask TSK object is 3. Within the for loop, this function gets a full buffer from the stream and prints the data in that buffer.

The DGN driver generates data "on demand," so the task does not block when it calls SIO\_get waiting for data. Since the streamTask object has a higher priority than the idle task, this task function runs to completion before the idle task can run to communicate instrumentation data to the host PC.



### Predicting the Result

**DSP/BIOS - IO**

In this example, the [doStreaming function](#) gets data from the stream and prints the data to a log.

What do you expect the results to be when you run the program? Click the arrow below to move to the next step.

Answer A	Answer B	Answer C
An error reading the buffer occurs because the buffer is empty.	The program gets the first buffer and prints the sine data to the log, but gets an error reading the log the next time through the loop.	The log lists a repeating sequence of 16 values ranging from -128 to 128 in a sine wave pattern in each buffer.

Answer A: An error occurs because the buffer is empty.

Answer B: An error occurs the second time through the loop because the buffer is empty.

Answer C: The log lists a repeating sequence of 16 values ranging from -128 to 128 in a sine wave pattern in each buffer.



### Testing With CCStudio

**DSP/BIOS - IO**

Run the program to see if your expectations were correct.

- Choose Project→Build or click the  (Incremental Build) toolbar button.

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007

2. Choose File→Load Program. Select the stotest1.out program you just built (usually in `C:\CCStudio_V3.10\myprojects\stotest\Debug`).
3. Choose DSP/BIOS→Message Log. Select trace in the Log Name list.
4. **Arrange and resize** the windows as needed to make them easy to see on your screen.
5. Run the program (Debug→Run).
6. When the program displays data in the Message Log, compare the results to your expectations.

Because the DGN driver generates data "on demand," there is always data available for SIO\_get in this program. So the correct answer in the previous step was Answer C.



### Things to Try

#### DSP/BIOS - 10

- Experiment with the properties of the DGN driver. Change the settings for the sine wave or cause the DGN driver to generate a random or constant value. Save your configuration file, rebuild the program, and look at the data in the trace log.
- The DGN driver generates data "on demand," so the task does not block when it calls SIO\_get waiting for data. How can you modify the program to allow the idle task, TSK\_Idle, to run before the streamtask has completed? [Answer](#)
- What causes the DGN driver to provide values? At what rate are those properties provided? [Answer](#)
- Compare the source code in stotest1.c and stotest2.c. Also, look at the objects that have been added to the configuration in stotest2.cdb. How are these versions of the program different? [Answer](#)
- Compare the source code in stotest2.c and stotest3.c. Also, look at the objects that have been added to the configuration in stotest3.cdb. How are these versions of the program different? [Answer](#)

#### See Also

To learn more about I/O handling, read Chapters 6 and 7 of the DSP/BIOS User's Guide, which is provided in PDF format. For reference information, see the DSP/BIOS Modules topic in the online help.

This concludes Managing Streams.



file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007

Getting Started With the Code Composer Studio Tutorial

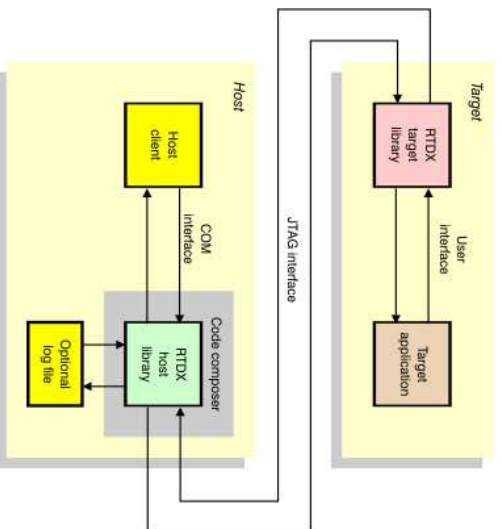
Page 430 sur 548

### Introduction

#### RTDX - Introduction

Real-Time Data Exchange (RTDX™) provides real-time, continuous visibility into the operations of target board applications. RTDX transfers data between a host computer and target devices without interfering with the target application. You may analyze and visualize the data on the host using the COM interface provided by RTDX. Clients such as Visual Basic™, Visual C++™, Excel™, LabView™, Matlab™, and others may easily use the COM interface. This realistic representation of your system operation may shorten development time.

RTDX forms a 2-way data pipe between a target application and a **host client** via a combination of hardware and software components, as shown below:



Data can be sent from the target application to the host client and vice versa. We can visualize this pipe as a collection of one or more thinner pipes, or channels, through which the data travels. This tags the data as belonging to a particular channel to distinguish various data. These channels are unidirectional; the data flows from the target application to the host client or vice-versa. Data can be input into a channel asynchronously, that is, at any time.

The target application sends data to the host by calling functions in the RTDX Target Library. These functions immediately buffer the data and then return. The RTDX Target Library then transmits the buffered data to the host without interfering in the target application. The host records the data into either a memory buffer or a RTDX log file, depending on the specified RTDX host recording mode. The recorded data can be retrieved by any client host application or the RTDX host interface. Windows™ platform computers provide a COM interface for the RTDX host interface.

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007

Similarly, a host client can send data to the target. The RTDX Host Library buffers all data sent to the target. If the RTDX Host Library receives a data request from the target application with sufficient data in the host buffer to satisfy the request, it sends the data to the target. The data is written to the requested location without interfering with the target application. The host notifies the RTDX Target Library upon operation completion.

This document assumes that you are familiar with your target processor, C programming, and Code Composer™.

For more information on available RTDX interfaces, see the RTDX Online Help.

System Requirements: Before you begin using this tutorial, make sure that your system has the following components:

- A target processor
- Microsoft™ Visual Basic™ or Microsoft Office™ installed on your PC

**Tip:** On most targets, RTDX utilizes interrupts to accomplish the data transfer. Therefore, RTDX performance may degrade during long lockout periods. Be aware that certain compiler options may also affect RTDX performance. For example, the C6x compiler interrupt threshold option, `-mic<n>`, can be used to limit the length of interrupt lockouts in compiled code. For more information, see the [TMS320C6000 Optimizing Compiler User's Guide \(SPRUJ87\)](#).

This tutorial module contains the following lessons:

- RTDX Configuration Considerations
- Using RTDX From Target Application
- Writing RTDX Host Clients
- Advanced RTDX Topics
- Example Codes Reference Library



## Overview

RTDX - L1

RTDX provides ActiveX Controls as Code Composer plugins to configure and control RTDX graphically, set up RTDX channels, and run diagnostic tests on RTDX.

This lesson introduces you to the RTDX plugins.

### Learning Objectives:

- Configure RTDX using channels and diagnostics

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007

## Getting Started With the Code Composer Studio Tutorial

Page 432 sur 548

- Identify the interrupt, unconfigured, test, uses
- Reconfigure the host buffer
- Identify the different modes of operation
- Record and review RTDX data in a log file

### Example: sect\_3

Application Objective:

This lesson uses various examples to demonstrate the options available with the RTDX tool. The final code documents for each example in this lesson are available for review in the Example Codes Reference Library in the Table of Contents under RTDX.

This lesson consists of the following steps:

- Configuring RTDX Graphically
- Setting Up RTDX Channels
- Running Diagnostics on RTDX
- Reconfiguring the Host Buffer
- Selecting the RTDX Mode of Operation
- Recording RTDX Data in a Log File
- Viewing a Pre-Existing Log File of Data



The forward arrow will take you to the next page in this lesson.

## Configuring RTDX Graphically

RTDX - L1

RTDX provides the RTDX Configuration Control as a Code Composer plugin to configure and control RTDX graphically. The RTDX Configuration Control is the main RTDX window. It allows you to view the current RTDX configuration settings, and enable or disable RTDX.

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

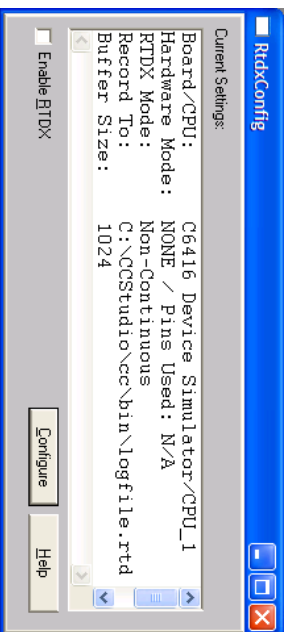
26/09/2007



**Note:** RTDX must be enabled when you run your application code. You must disable RTDX at all other times.

To configure RTDX:

1. In the Code Composer Studio™ IDE, select Tools→RTDX→Configuration Control.



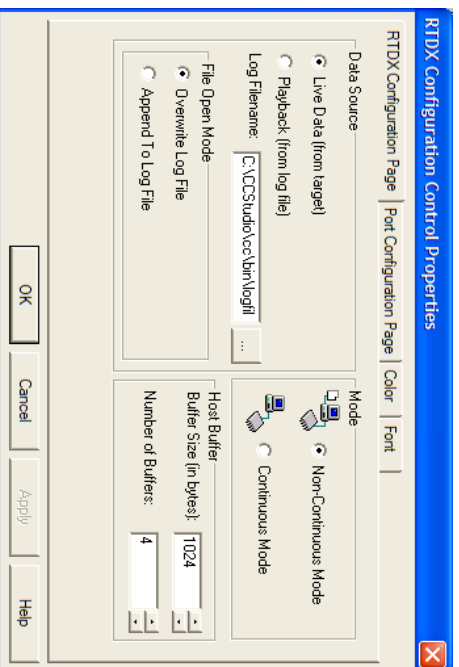
2. To enable RTDX, check the Enable RTDX checkbox; click again to disable RTDX.
  3. If not disabled by default, [disable RTDX](#).
  4. Click the Configure button to access the RTDX Configuration Control Properties page.
- By default, it displays the RTDX Configuration Page tab.

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007

Getting Started With the Code Composer Studio Tutorial

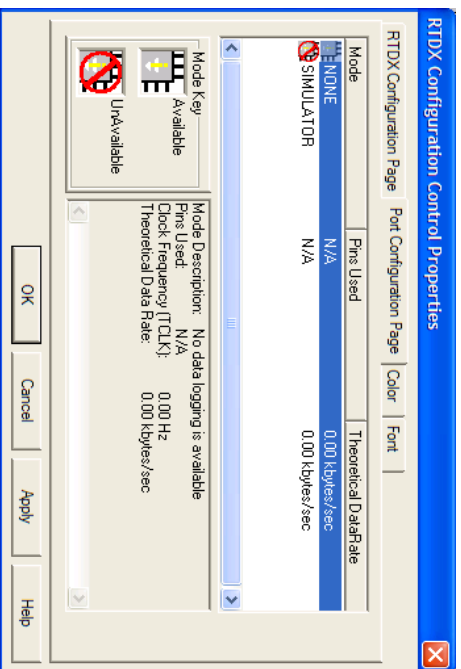
Page 434 sur 548



5. Reconfigure RTDX by configuring the following:
  - **Data Source**
  - RTDX Mode of Operation: This is discussed later in this section.
  - **File Open Mode**
  - **Host Buffer:** This is discussed later in this section.
6. Click the Port Configuration Page tab to display the Port Configuration Page.

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007



7. If you wish to change your port, select the desired port configuration mode. Otherwise, leave it in default mode.  
The various emulation technologies are represented as various port configuration modes in the Port Configuration Page. The RTDX Configuration Control determines which mode is available to be used at the time a target application is loaded or when the control is first brought up. Modes that are not available have a red slash on the associated image.
8. Click OK for configurations to take effect.
9. In the Configuration Control window, click the Enable RTDX checkbox to enable RTDX.



### Viewing RTDX Channels

RTDX - L1

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, and enable or disable channels on the list.

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007

### Getting Started With the Code Composer Studio Tutorial

Page 436 sur 548

#### To access the RTDX Channel Viewer Control:

Select Tools →RTDX→Channel Viewer Control.



To remove a channel:

1. There should be a list of channels. Select the channel you want to remove.
2. Right-click on the channel to display the channel menu.
3. Select Delete Channel to remove the channel from the viewable list.

To add a channel:

1. Right-click in the viewable list to display the context menu.
2. Select Add Channel.
3. In the Channel Name field, enter the name of the channel to add.
4. Click OK. The channel name appears in the viewable list.

To enable or disable a channel once it has been added:

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007

1. In the Viewable list, select the channel you want to enable or disable.
2. Right-click on the channel to display the channel context menu.
3. Select Enable/Disable Channel.

The following table shows the channel states in the Viewable list:

- Channel is currently enabled.
- Channel is currently disabled.
- Channel is unknown.

When a channel is enabled, any data written to the channel is sent to the host. Channels are initialized to be disabled.



### Running Diagnostics on RTDX

RTDX - L1

RTDX provides the RTDX Diagnostics Control as a plugin to verify RTDX operation. The diagnostic tests test the basic functionality of target-to-host transmission and host-to-target transmission.

The Internal Test simulates the receipt of data from a target application. Running this test ensures that Code Composer can process data correctly. There is no associated target application with this test.

The Target-To-Host Test validates the target's ability to transmit data to the host and the host's ability to receive data from the target. This test uses the target application t2h.out. The path for the t2h.out file is [C:\CCStudio\\_v3.10\examples\Target\\_Vtdx\T2h](file:///C:/CCStudio_v3.10/examples/Target_Vtdx\T2h).

The Host-To-Target Test validates the target's ability to receive data from the host and the host's ability to transmit data to the target. This test uses the target application h2t.out. The path for the h2t.out file is [C:\CCStudio\\_v3.10\examples\Target\\_Vtdx\H2t](file:///C:/CCStudio_v3.10/examples/Target_Vtdx\H2t).

To run a diagnostic test on RTDX:

1. From the Tools menu, select RTDX→Configuration Control.

<file:///C:/Documents and Settings/Pierre-Amand\Local Settings\Temp\~hhBA53.htm>

26/09/2007

### Getting Started With the Code Composer Studio Tutorial

Page 438 sur 548

2. Click the Enable RTDX checkbox to enable RTDX.
3. From the Tools menu, select RTDX→Diagnostics Control.



4. Select one of the following tests:

- Internal Test
- Target-To-Host Test
- Host-To-Target Test

5. Click the Run Test button to run the test you selected.  
The test results appear in the status window upon completion.
6. Click the Halt Test button to stop the test from running before it completes.



### Reconfiguring the Host Buffer

RTDX - L1

Host buffer size is the size in K-bytes of the RTDX Host Library's main buffer. The main buffer stores one message from the target application. This buffer must be larger than the largest message from the target application. For 32-bit target architectures, the buffer must be 8 bytes larger. For 16-bit architectures, it must be 4 bytes larger.

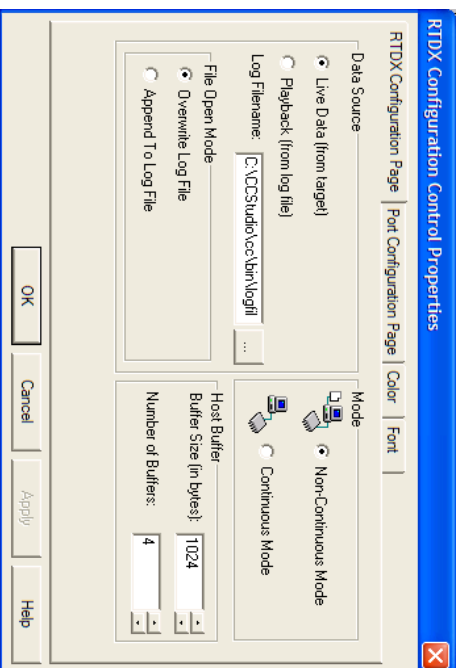
<file:///C:/Documents and Settings/Pierre-Amand\Local Settings\Temp\~hhBA53.htm>

26/09/2007

Reconfigure the host buffer on if you receive the following error message via the COM API method `GetStatusString()`: "A data message was received which cannot fit into the buffer allocated on the host. To avoid data loss, reconfigure the buffer and run the application again."

To reconfigure the host buffer, follow these steps:

1. In Code Composer, select Tools→RTDX→Configuration Control.
2. If not disabled by default, [disable RTDX](#).
3. Click the Configure button. By default, the RTDX Configuration Page tab is displayed.



4. In the Buffer Size (in bytes) field, enter the desired buffer size. By default, the Main Buffer Size is set to 1024 bytes.
5. In the Number of Buffers field, enter the desired number of buffers. By default, the number of buffers is set to 4.
6. Click OK for configurations to take effect.
7. In the Configuration Control window, click the Enable RTDX checkbox to enable RTDX.

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007

Getting Started With the Code Composer Studio Tutorial

Page 440 sur 548



### Selecting the RTDX Mode of Operation

RTDX - LI

The RTDX Host Library provides two modes of receiving data from the target application. These are:

- Non-Continuous mode
- Continuous mode

In non-continuous mode, data received from the target is written to a log file (\*.rtd) on the host. The data recorded in this mode can be played back at a later time.

In continuous mode, data received from the target is recorded into a circular memory buffer by the RTDX Host Library. It is not written to a log file (\*.rtd) on disk. In continuous mode, the host client must continually read from each output channel in order to drain the host buffer of data. Failure to do this may result in the host buffers becoming full. When the host buffers become full the RTDX Host Library requests the RTDX Target Library to stop sending data to the host.

If the RTDX Target Library is unable to drain the target buffer of data and the target application continues to call `RTDX_write`, the target buffer eventually becomes full and is unable to hold more data. If the target buffer is full, calls to `RTDX_write` return a failure.

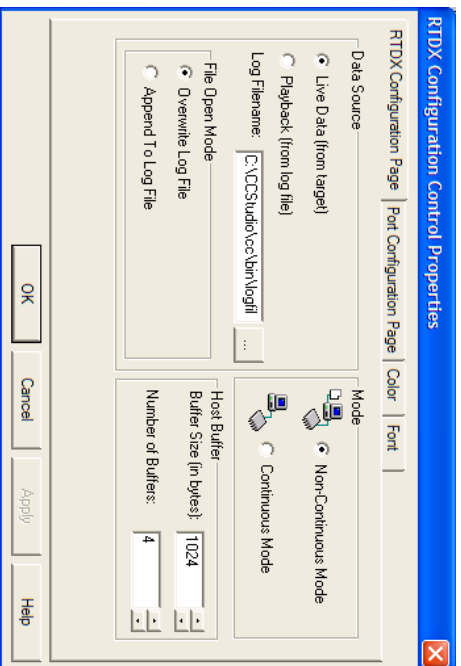
The flow of data from the target resumes once the host clients have consumed enough data from the host buffer for the RTDX Host Library to request the RTDX Target Library to resume sending data. In addition, the host client must open all target output channels upon startup to avoid data loss to any of the channels.

To select the desired RTDX host recording mode, follow these steps:

1. In Code Composer Studio, select Tools→RTDX→Configuration Control.
2. If not disabled by default, [disable RTDX](#).
3. Click the Configure button to open the RTDX Configuration Control Properties page. By default, the RTDX Configuration Page tab is displayed.

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007



4. Select the desired RTDX Mode of operation from the following options:

- Non-Continuous Mode: Select this mode when you want to capture a finite amount of data and record it in a log file (\*.rtd). The following lesson shows you how to record RTDX data in a log file.

**Caution:** Do not attempt to save a log file over a network connection as it may violate real-time constraints.

- Continuous Mode: Select this mode when you want to continuously obtain and display the data from a target application, and you do not need to store the data in a log file. Use this mode to continuously retrieve data from the target.

5. Click OK for configurations to take effect.

6. In the RTDX→Configuration Control window, click the Enable RTDX checkbox to enable RTDX.



file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007

Getting Started With the Code Composer Studio Tutorial

Page 442 sur 548

### Recording RTDX Data in a Log File

RTDX - 11

RTDX allows you to record data received from the target to a log file (\*.rtd) on the host. The recorded data can be viewed at a later time using the Playback mode.

You can specify the name of the log file using the RTDX Configuration Control window or by calling the RTDX\_CDM method `ConfigureLogFile()`. In this lesson, you learn how to specify the name of the log file using the RTDX Configuration Control window. The data from the target can only be recorded when RTDX has been configured to operate in non-continuous mode.


This lesson shows how to record integer values 1 to 10 in a log file.

Before you begin, [disable RTDX](#).

To record data in a log file, follow these steps:

1. From the Project menu in Code Composer, browse to the `C:\CCStudio_v3.10\tutorial\target\sect_3\less_4` directory and select `S314.pjt`. If you configured your target for big endian, select `S314_e.pjt`.
2. Click Open.
3. From the Project menu, select Build to build your example target application executable, `S314.out`.

**Note:** Be aware of compiler options that may affect RTDX performance. For example, the `C6x` compiler interrupt threshold option, `-mi<n>`, can be used to limit the length of interrupt lockouts in compiled code. For more information, see the [TMS320C6000 Optimizing Compiler User's Guide \(SPRU187\)](#).

4. From the Code Composer Studio™File menu, select Load Program.
5. In the Load Program dialog box, browse to the `C:\CCStudio_v3.10\tutorial\target\sect_3\less_4\Debug` directory and select `S314.out`.
6. Click Open.
7. From the Tools menu, select RTDX→Configuration Control.
8. Click the Configure button to open the RTDX Configuration Control Properties page. The RTDX Configuration Page is displayed by default.
9. In the RTDX Configuration Page, set the RTDX Mode of operation to Non-Continuous Mode.
10. Select Live Data (from target) as the Data Source.
11. In the Log Filename field, click  to open the Save a log file dialog box.
12. In the Filename field, enter the name of the log (\*.rtd) file you wish to save, and the location to which you want to save it. For this lesson, call this file `S314.rtd`, and save your log file

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007

in the C:\CCStudio\_v3.10\tutorial\target\sect\_3\less\_4\Debug directory.

**Caution:** Do not attempt to save a log file over a network connection as it may violate real-time constraints.

13. Click Save.
14. Click OK.
15. In the Configuration Control window, check the Enable RTDX checkbox to enable RTDX.
16. From the Debug menu, select Run to run the target application.
17. View the log file generated. The following lesson will explain this in more detail.
18. From the Project menu, select Save to save your project.
19. Close Code Composer Studio™.



### Viewing a Pre-Existing Log File of Data

RTDX - L1

RTDX allows a host client to retrieve data from a pre-recorded log file (\*.rtd). To use this feature, you must configure RTDX to operate in Playback mode. You can configure RTDX to operate in Playback mode from the RTDX Configuration Control window or by calling the methods of the RTDX COM interface. In this lesson, you learn how to configure RTDX to operate in Playback Mode using the RTDX Configuration Control.

This lesson shows how to view integer values 1 to 10 recorded in the log file in Lesson S3L4, Recording RTDX Data in a Log File.

To view a pre-existing log file of data, follow these steps:


1. To restore your project from the previous lesson, start Code Composer Studio.
2. From the Project menu in Code Composer, select Recent Project Files and choose S3L4.pjt.
3. From the File menu in Code Composer, select Load Symbol... to open the Load Symbol dialog box.
4. Browse to the C:\CCStudio\_v3.10\tutorial\target\sect\_3\less\_4\Debug directory and select S3L4.out.

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007

### Getting Started With the Code Composer Studio Tutorial

Page 444 sur 548

5. Click Open.
6. In Code Composer, select Tools→RTDX→Configuration Control.
7. [Disable RTDX](#).
8. Click Configure to open the RTDX Configuration Control Properties page. By default, the RTDX Configuration Page is displayed.
9. In the RTDX Configuration Page, set the RTDX Mode of operation to Non-Continuous Mode.
10. Select Playback (from log file) as the Data Source.
11. In the Log Filename field, click  to open the log file dialog box.
12. Browse to the C:\CCStudio\_v3.10\tutorial\target\sect\_3\less\_4\Debug directory.
13. Select the pre-recorded log file S3L4.rtd or enter S3L4.rtd in the Filename field and click Open.
14. Open an MS-DOS™ Prompt window and type C:\CCStudio\_v3.10\tutorial\target\sect\_3\less\_4\S3L4.exe to invoke the example executable host client.

Numbers ranging from 1 to 10 appear in the MS-DOS Prompt window, verifying that the host client has read the contents of the log file, S3L4.rtd.



### Overview

RTDX - L2

RTDX includes a target and a host library for different functions. The target application sends data to the host by calling functions in the RTDX Target Library. These functions immediately buffer the data and then return. The RTDX Target Library then transmits the buffered data to the host via the JTAG interface without interfering with the target application. The host records the data into either a memory buffer or a RTDX log file, depending on the host recording mode specified. The recorded data can be retrieved by any host application using the RTDX host interface.

Alternatively, your *host client* can send integer data to or receive integer data from the target application using the COM interface. A host client is any application that is capable of utilizing the COM interface. All data to be sent to the target is written to a memory buffer within the RTDX Host Library. When the RTDX Host Library receives a request for data from the target application, the data in the host buffer is sent to the target via the JTAG interface. The data is written to the requested location on the target without interfering with the target application. The host notifies the RTDX Target Library when the operation is complete.

In a similar way, any host application that is a host client can retrieve target-sent data recorded into a memory buffer or a RTDX log file.

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007

In this lesson, you learn how to use RTDX to send or receive various types of data from the host. You also write RTDX host clients that can send or receive various types of data from the target.

Prerequisite: [Developing a Simple Program](#)

**Learning Objectives:**

- Identify the code requirements for using RTDX
- Identify the operations specified by the code
- Send and receive information from the host
- Reconfigure the target buffer

**Examples: Various in sect\_1 and sect\_2**

**Application Objective:**

This lesson uses sect\_1 examples to show you how to use RTDX to send various types of data to the host, as well as receive various types of data from the host. You learn what code to insert into these examples to build a target application that can send data to and receive data from the host. Finally, you learn how to reconfigure the target buffer to accommodate the quantity of data being sent to the host. The final code documents for each example in this lesson are available for review in the Example Codes Reference Library in the Table of Contents under RTDX.

This lesson consists of the following steps. To go directly to a step, click on the link below:

[Sending an Integer to the Host](#)

[Receiving an Integer From the Host](#)

[Reconfiguring the Target Buffer](#)

[In-Depth Topics](#)



The forward arrow will take you to the next page in this lesson.

**[Sending an Integer to the Host](#)**

RTDX - L2

The purpose of this lesson is to use RTDX to send a single integer to the host, using lesson SLL1.pjt. Over the course of the lesson, you will learn about the following types of files:

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007

**Getting Started With the Code Composer Studio Tutorial**

Page 446 sur 548

- .c This file contains source code that provides the main functionality of this project
- .h This file declares the buffer C-structure as well as define any required constants
- .pjt This file contains all of your project build and configuration options

Sending data to the host requires a few basic steps:

- Open your project and view your application code in Code Composer Studio™ IDE.
- Insert specific RTDX syntax in your application code to allow real-time data transfer.
- Process the data and test your application code.

Before you begin, [disable RTDX](#).

To open your project and modify the application code, follow these steps:

1. If you have not already done so, from the Windows Start menu, choose Programs→Texas Instruments→Code Composer Studio 3.1→Code Composer Studio 3.1. (Or, double-click the Code Composer Studio icon on your desktop.)
2. From the Project menu, select Open.
3. Browse to the C:\CCStudio\_v3.1.0\tutorial\target \sect\_1\less\_1 directory, and select SLL1.pjt and click Open. If you configured your target for big endian, select SLL1\_e.pjt.
4. If the Project View window is not open, click on View→Project.
5. In the project view, expand the Project list by clicking the + signs next to Projects, SLL1.pjt, Source, and open the SLL1.c file.

6. The prototypes for the RTDX target interface are defined in the header file, rtdx.h, and must be added to your target application. You should insert the RTDX header file code at the beginning of the file:

```
#include <rtdx.h>
```

Please see the Example Target Application C Code SLL1.c [link](#) to view the modified file.

7. RTDX channels are data structures in target memory that must be declared as global data objects. The name chosen for a channel is arbitrary and dependent upon your needs. The example in this lesson uses the name ochan to define an output channel. To declare a global output channel for sending (writing) data to the host, insert the following after the header code:

```
RTDX_CreateOutputChannel( ochan );
```

Please see the Example Target Application C Code SLL1.c [link](#) to view the modified file.

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007

- Initialize target using the macro TARGET\_INITIALIZE() or insert your own code. However, skip this step in this tutorial, as the TARGET\_INITIALIZE() macro has already been defined in the examples. Please see the Example Target Application C Code SLL3.c [link](#) to view the modified file. To initialize the target normally, you would insert the following code:

```
TARGET_INITIALIZE();
```

**Note:** You must initialize the target if your processor has a different initialization sequence needed for RTDX to function properly, such as enabling interrupts. The examples used in this tutorial are target independent, as the target-specific initialization has been defined in the macro TARGET\_INITIALIZE(). This macro is defined as part of the example and is not specific to RTDX itself.

- RTDX channels are initialized as disabled. For actual data transfer to occur, you must enable the RTDX channel. Channels are enabled by either calling a RTDX target function, as shown in this example, or remotely via the debugger, discussed in the next section. To enable the output channel for writing, insert the following code after the target initialization code:

```
RTDX_enableOutput( kochan );
```

Please see the Example Target Application C Code SLL1.c [link](#) below to view the modified file.

- You must then send (write) an integer to the host by specifying the following code:

```
status = RTDX_write( kochan, sizeof(data) );
```

Please see the Example Target Application C Code SLL1.c [link](#) to view the modified file.

- Finally, you must disable the RTDX channel to stop data from transferring. A call to a RTDX target function using a disabled channel reports success but does not actually transfer any data. To disable the output channel from sending (writing) data, insert the following code before the Program Complete code:

```
RTDX_disableOutput( kochan );
```

**Example Target Application C Code SLL1.c** (the inserted code is in red)

To process the data and test your application code, follow these steps:

- From the File menu, select Save to save your example source file, SLL1.c.
  - From the Project menu, select Build to build your example target application executable, SLL1.out.
- Note:** Be aware of compiler options that may affect RTDX performance. For example, the C6x compiler interrupt threshold option, `-mi<n>`, can be used to limit the length of interrupt lockouts in compiled code. For more information, see the [TMS320C6000 Optimizing Compiler User's Guide \(SPRU187\)](#).
- From the File menu, select Load Program.
  - In the Load Program dialog box, browse to the `C:\CCStudio_v3.10\tutorial\target\sect_1\less_1\SLL1` directory and select SLL1.out.

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007

Getting Started With the Code Composer Studio Tutorial

Page 448 sur 548

- Click Open.
  - In the main menu, select Tools→RTDX→Configuration Control.
  - In the Configuration Control window, click inside the Enable RTDX checkbox to enable RTDX.
  - From the Debug menu, select Run to run your application to completion.
  - Open an MS-DOS™ Prompt window and type `C:\CCStudio_v3.10\tutorial\target\sect_1\less_1\SLL1.exe` to invoke the example executable host client. The number 5 appears in the MS-DOS Prompt window, verifying that the host client has read integer value 5 sent by the target application.
- Before you move to the next lesson, [disable RTDX](#), and close the SLL1 project and all open windows in CCS.



### Receiving an Integer From the Host

**RTDX - L2**

The purpose of this lesson is to learn how to use RTDX to receive a single integer from the host.

The target application requests data from the host and waits for its arrival with one function call to RTDX\_read. The target requests data from the host by sending a special read request message to the host. Later, when the host client has supplied the data, the RTDX Host Library satisfies the request by writing the requested quantity of data to the address specified by the target application.

Before you begin, [disable RTDX](#).

To open your project and view the application code:

- From the Project menu, select Open.
- Browse to the `C:\CCStudio_v3.10\tutorial\target\sect_1\less_3` directory, select SLL3.pjt and click Open. If you configured your target for big endian, select SLL3\_e.pjt.
- If the Project View window is not open, click on View→Project.
- In the project view, expand the Project list by clicking the + signs next to Projects, SLL3.pjt, Source, and open the SLL3.c file.
- First you must declare a global input channel for receiving (reading) data from the host by inserting the following code into the SLL3.c file after the `#include <stdio.h> /* C_LJ0 */` instruction.

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007



```
RTDX_CreateInputChannel( &chan );
```

The example in this lesson uses the arbitrary channel name `ichan` to define an input channel. Please see the [Example Target Application C Code S1L3.c link](#) to view the modified file.

- RTDX channels are initialized to be disabled. Therefore, the channel must be enabled before the application requests data from it. To enable the input channel for reading, place the following code after the `TARGET_INITIALIZE()` call. Please see the [Example Target Application C Code S1L3.c link](#) to view the modified file.

```
RTDX_enableInput( &ichan );
```

- Next, you will insert a request and wait to receive data with one function call to `RTDX_read` inserted after the previously inserted code. Please see the [Example Target Application C Code S1L3.c link](#) to view the modified file.

```
status = RTDX_read( &chan, &data, sizeof(data) );
```

- Finally, you will disable the input channel from receiving (reading) data by inserting the following data after the `printf("Value %d was received from the host\n", data);` function. To stop data from transferring, disable the RTDX channel. A call to a RTDX target function using a disabled channel reports success but does not actually transfer any data. Please see the [Example Target Application C Code S1L3.c link](#) to view the modified file.

```
RTDX_disableInput( &ichan );
```

### [Example Target Application C Code S1L3.c](#)

To process the data and test your application code, follow these steps:

- From the File menu, select Save to save your example source file, `S1L3.c`.
- From the Project menu, select Build to build your example target application executable, `S1L3.out`.

**Note:** Be aware of compiler options that may affect RTDX performance. For example, the C6x compiler interrupt threshold option, `-mi<n>`, can be used to limit the length of interrupt lockouts in compiled code. For more information, see the [TMS320C6000 Optimizing Compiler User's Guide \(SPRU187\)](#).

- From the File menu, select Load Program.
- In the Load Program dialog box, browse to the `C:\CCStudio_v3.10\tutorial\target\sect_1\less_3\S1L3` directory and select `S1L3.out`.
- Click Open.
- In Code Composer, select Tools-->RTDX-->Configuration Control.
- In the Configuration Control window, click inside the Enable RTDX checkbox to enable RTDX.

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007

## Getting Started With the Code Composer Studio Tutorial

Page 450 sur 548

- From the Debug, select Run, and run your application, it will not halt until you have sent the message from the host.
- Open an MS-DOS™ Prompt window.

- Type `C:\CCStudio_v3.10\tutorial\target\sect_1\less_3\S1L3.exe` in the prompt window to invoke the example executable host client.

The message, "Value 5 was received from the host, Program Complete!" appears in the Code Composer Standard Output (Stdout) window, verifying that the target application has received integer value 5 sent by the host client.

Before you move to the next lesson, [disable RTDX](#), and leave the project open.



### Reconfiguring the Target Buffer

RTDX - L2

All data to be sent to the host is first recorded into a circular buffer declared within the RTDX Target Library. This allows RTDX\_write to immediately return control to the target application while the buffered data is sent to the host. As the buffered data is sent to the host, it is removed from the buffer, thereby, clearing room for the next message written by the target application.

The messages sent to the host to request data are also initially recorded in the target buffer. However, data arriving from the host is written directly to the requested location and is not written to the target buffer.

The RTDX target buffer must be large enough to hold the largest possible message written by the target. The size of the target buffer may be increased or decreased to accommodate the quantity of data being sent to the host. There is also a RTDX host buffer into which these messages are received. If the size of the RTDX target buffer is increased to accommodate a large message, the RTDX host buffer may also need to be increased.

At this time, there is no target buffering in the RTDX Simulator target libraries. Therefore, there is no need to resize a target buffer for large messages when using the simulator. However, the host buffer size may still need to be increased to accommodate large messages.

To reconfigure the target buffer size, follow these steps:

- Make a copy of the default `rdx_buf.c` file. The path for this file is `C:\CCStudio_v3.10\c6000\rtdx\lib\rdx_buf.c`.
- From the Project menu, select Add Files to Project....
- In the dialog box, browse to the copy of `rdx_buf.c` and add it to the Code Composer Studio™ project.
- In the copy of `rdx_buf.c`, change the value of `BUFRSZ` to the desired value.

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007

- From the Project menu, select Build to compile and link the application with the modified rdx\_buf.c.

**Note:** You must compile your copy of rdx\_buf.c to match the RTDX Target Library you are linking with.

This concludes the Using RTDX From Target Application lesson. For further tutorials on using RTDX from the target application, please click on the In-Depth Topics book in the Table of Contents menu and choose a lesson, or follow these links:

[Sending an Array of Integers to the Host](#)

[Receiving an Integer from the Host Asynchronously](#)

[Receiving an Array of Integers from the Host](#)

Before you move to the next lesson, [disable RTDX](#), and close the SLI3 project and all open windows in CCS.



---

## **Sending an Array of Integers to the Host**

### **RTDX - L2**

The purpose of this lesson is to learn how to use RTDX to send (write) an entire array of integers from the target to the host. The example used in this lesson shows how to send integer values 1 to 10 to the host.

To send data to the host, you must follow this basic procedure:

- Open your project and view your application code in Code Composer.
- Insert specific RTDX syntax in your application code to allow real-time data transfer.
- Process the data and test your application code.

Before you begin, [disable RTDX](#). Open SLI2.pjt in Code Composer and view the example application code, SLI2.c, as per the instructions specified in the introductory lessons.

In your application code:

You must send (write) an entire array of integers to the host with one function call to RTDX\_write. In your application code, insert the following code after the enable channel instructions. Please see the Example Target Application C Code SLI3.c [link](#) to view the modified file:

```
status = RTDX_write( kochan, arraydata, sizeof(arraydata) );
```

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007

Getting Started With the Code Composer Studio Tutorial

Page 452 sur 548

[Example Target Application C Code SLI2.c](#)

To process the data and test your application code, follow these steps, as specified in the previous introductory lessons:

- Save and build your target application to produce the target application executable, \*.OUT.
- Load your target application executable.
- Enable RTDX and run your target application.
- Run the host client.

Numbers ranging from 1 to 10 appear in the MS-DOS prompt window, verifying that the host client has read integer values 1 to 10 sent by the target application.

Before you move to the next lesson, [disable RTDX](#), and close the SLI2 project and all open windows in CCS.



---

## **Receiving an Integer From the Host Asynchronously**

### **RTDX - L2**

The related introductory lesson, [Receiving an Integer From the Host](#), covered how to use a blocking read to request data from the host. This lesson describes how to issue a non-blocking read, a type of read request that does not wait for the data to arrive. Instead, control is immediately returned to the target application after sending the read request to the host.

The advantage of non-blocking reads is that the target application can perform other tasks while the data is being received from the host. This is what is known as asynchronous behavior.

Additional functions in the target API are provided to allow you to test when the data arrives and the amount of data that arrives on the target. Only one outstanding read request is allowed per input channel. After issuing a read, a channel state becomes busy. The channel returns to the idle state only after the data arrives. Any attempt to issue successive read requests on an input channel that is already busy returns a failure to the target application.

The example used in this lesson shows how to receive integer value 5 from the host asynchronously.

To receive data from the host, you must follow this procedure:

- Open your project and view your application code in Code Composer.
- Insert specific RTDX syntax in your application code to allow real-time data transfer.
- Process the data and test your application code.

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007

Before you begin, [disable RTDX](#). Open S114.pjt in Code Composer Studio and view the example application code, S114.c, as per the instructions specified in the introductory lessons.

In your application code:

1. First you must request data from the host by issuing a non-blocking read call. Insert the following code after the `RTDX_enableInput( &chan );` instruction. Please see the Example Target Application C Code S114.c [link](#) to view the modified file.

```
status = RTDX_readNB( &chan, &data, sizeof(data) );
```
2. When using a non-blocking read request, the target application must synchronize with the arrival of the requested data before attempting to use it. Synchronize target code with arrival of data by inserting the following code after the `while ( status ) {` instruction. Please see the Example Target Application C Code S114.c [link](#) to view the modified file.

```
status = RTDX_channelBusy( &chan );
```
3. It is possible for the host to partially satisfy the target's request for data. Hence, verify all requested data. Verify that your program has received all requested data by inserting the following code after the `RTDX_Poll();#endif }` instruction. Please see the Example Target Application C Code S114.c [link](#) to view the modified file.

```
status = RTDX_sizeofInput( &chan );
```

#### [Example Target Application C Code S114.c](#)

To process the data and test your application code, follow these steps, as specified in the previous introductory lessons:

- [Save and build your target application to produce the target application executable, \\*.out.](#)
- [Load your target application executable.](#)
- [Enable RTDX and run your target application.](#)
- [Run the host client.](#)

The message "Value 5 was received from the host," appears in the Code Composer Standard Output (Stdout) window, verifying that the target application has received integer value 5 sent by the host client.

Before you move to the next lesson, [disable RTDX](#), and close the S114 project and all open windows in CCS.



[file:///C:/Documents and Settings/Pierre-Armand/Local Settings/Temp/~hhBA53.htm](#)

26/09/2007

Getting Started With the Code Composer Studio Tutorial

Page 454 sur 548

## Receiving an Array of Integers From the Host

### RTDX - L2

The purpose of this lesson is to write a RTDX target application that can receive an array of integers from the host. Similar to write operations, it is possible to request a block of data from the host. This is available for both the blocking and non-blocking forms of read.

The example used in this lesson shows how to receive integer values 1 to 10 from the host with a blocking read request.

To receive data from the host, you must follow this procedure:

- Open your project and view your application code in Code Composer.
- Insert specific RTDX syntax in your application code to allow real-time data transfer.
- Process the data and test your application code.

Before you begin, [disable RTDX](#). Open the S115.pjt project in Code Composer Studio and view the example application code, S115.c, as per the instructions specified in the introductory lessons.

In your application code:

You must request and wait to read data from the host by issuing a blocking read call inside the loop counter. Insert the following code after the `RTDX_enableInput( &chan );` instruction. Please see the Example Target Application C Code S115.c [link](#) to view the modified file.

```
status = RTDX_read( &chan, arraydata, sizeof(arraydata) );
```

#### [Example Target Application C Code S115.c](#)

To process the data and test your application code, follow these steps, as specified in the previous introductory lessons:

- [Save and build your target application to produce the target application executable, \\*.out.](#)
- [Load your target application executable.](#)
- [Enable RTDX and run your target application.](#)
- [Run the host client.](#)

Ten lines of output with numbers ranging from 1 to 10 appear in the Code Composer Standard Output (Stdout) window, verifying that the target application has received integer values 1 to 10 sent by the host client.

[file:///C:/Documents and Settings/Pierre-Armand/Local Settings/Temp/~hhBA53.htm](#)

26/09/2007

Before you move to the next lesson, [disable RTDX](#) , and close the S115 project and all open windows in CCS.



---

## Overview

### RTDX - L3

Your **host client** can send integer data to or receive integer data from the target application using the COM interface. A host client is any application that is capable of utilizing the COM interface.

All data to be sent to the target is written to a memory buffer within the RTDX Host Library. When the RTDX Host Library receives a request for data from the target application, the data in the host buffer is sent to the target via the JTAG interface. The data is written to the requested location on the target without interfering with the target application. The host notifies the RTDX Target Library when the operation is complete.

In a similar way, any host application that is a host client can retrieve target-sent data recorded into a memory buffer or a RTDX log file.

In this lesson, you learn how to write RTDX host clients that can send various types of data to the target, as well as receive various types of data from the target.

#### Prerequisite: Using RTDX From Target Application

#### Learning Objectives:

- Send an Integer to the Target
- Receive an Integer From the Target
- Remotely Enable and Disable a Channel

#### Example: sect\_2

#### Application Objective:

In this lesson, you learn how to write RTDX host clients. This lesson uses sect\_2 examples to write RTDX host clients that send various types of data to the target, as well as receive various types of data from the target. You learn what code to insert into these examples to build RTDX host clients that can send data to and receive data from the target. The final code documents for each example in this lesson are available for review in the Example Codes Reference Library in the Table of Contents under RTDX.

This lesson consists of the following steps:

- [Receiving an Integer From the Target](#)
- [Sending an Integer to the Target](#)

[file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm](#)

26/09/2007

---

## Getting Started With the Code Composer Studio Tutorial

Page 456 sur 548

[Remote Enabling and Disabling of a Channel](#)

[In-Depth Topics](#)



The forward arrow will take you to the next page in this lesson.

---

## Receiving an Integer From the Target

### RTDX - L3

The purpose of this lesson is to learn how to apply the knowledge acquired in the previous lesson to write a simple RTDX host client that reads single integers from the target application until there is no more data available. The example used in this lesson shows how to receive integer value 5 from the target.

To receive data from the target, you must follow this procedure:

- Open your application code in either a Visual Basic or an Excel project.
- Insert specific RTDX syntax in your application code to allow real-time data exchange.
- Process the data and test your application code.

Before you begin, [disable RTDX](#) .

1. Open the example application code, S2L2.bas, in either a Visual Basic or an Excel project.

The path for the example source file, S2L2.bas, is [C:\CCStudio\\_v3.10\tutorial\target\sect\\_2\less\\_2](#). Import the source file into either a Visual Basic or an Excel project and view the application code. For information on importing files, see the Microsoft Visual Basic or the Excel Help.

2. First you must define and include the RTDX return code constants by inserting the following code in the beginning of the file. Please see the Example Visual Basic Code S2L2.bas [link](#) to view the modified file.

```
Const Success = &H0 ' Method call is valid
Const Failure = &H80004005 ' Method call failed
Const ENODATAavailable = &H8003001E ' No data was available.
' However, more data may be
' available in the future.
Const ENDOfLogFile = &H80030002 ' No data was available.
' The end of the log file has
' been reached.
```

[file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm](#)

26/09/2007

3. Declare a variable of type Object after the Sub main() instruction. Please see the Example Visual Basic Code S2L2.bas [link](#) to view the modified file.  

```
Dim rtdx As Object
```
4. Create an instance of the RTDX COM object after the On Error GoTo Error\_Handler instruction. Please see the Example Visual Basic Code S2L2.bas [link](#) to view the modified file.  

```
Set rtdx = CreateObject("RTDX")
```
5. Open a channel for reading under the previous instruction. The example in this lesson uses the arbitrary channel name ochan to define an output channel. Please see the Example Visual Basic Code S2L2.bas [link](#) to view the modified file.  

```
status = rtdx.Open("ochan", "R")
```
6. Read data from the channel before the Select Case (status) instruction. Please see the Example Visual Basic Code S2L2.bas [link](#) to view the modified file.  

```
status = rtdx.Read14(data)
```
7. Close the channel from reading after Loop Until status = EEndOfFile. Please see the Example Visual Basic Code S2L2.bas [link](#) to view the modified file.  

```
status = rtdx.Close()
```
8. Release the reference to the RTDX COM object after the previous instruction. Please see the Example Visual Basic Code S2L2.bas [link](#) to view the modified file.  

```
Set rtdx = Nothing
```

#### [Example Visual Basic Code S2L2.bas](#)

Click [here](#) to look at the Visual C++ version of the Visual Basic code S2L2.bas.

**Note:** Before you proceed further, make sure that the Visual Basic Immediate window is visible.

To process the data and test your application code, follow these steps:

1. From the Project menu in Code Composer, select Open.
2. Browse to the C:\CCStudio\_v3.10\tutorial\target\sect\_2\less\_2 directory, select S2L2.pjt and click Open. If you configured your target for big endian, select S2L2\_e.pjt.
3. From the Project menu, select Build to build your example target application executable, S2L2.out.

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007

## Getting Started With the Code Composer Studio Tutorial

Page 458 sur 548

4. From the File menu, select Load Program.
5. In the Load Program dialog box, browse to the C:\CCStudio\_v3.10\tutorial\target\sect\_2\less\_2\Debug directory and select S2L2.out.
6. Click Open.
7. In Code Composer, select Tools->RTDX->Configuration Control.
8. In the Configuration Control window, click inside the Enable RTDX checkbox to enable RTDX.
9. From the Debug menu, select Run to run your application.
10. Run the Visual Basic code , S2L2.bas, in either Visual Basic or Excel. For updated information on running code, see the Microsoft Visual Basic or the Excel Help.

The message, "Value 5 was received from the target," appears in the Visual Basic Immediate window, verifying that the host client has received integer value 5 from the target.

Before you move to the next lesson, [disable RTDX](#) , and close the S2L2 project and all open windows in CCS.



### Sending an Integer to the Target

#### RTDX - L3

A client application can send data to the target application by writing data to the target. RTDX client applications write data to the target indirectly. Data sent from the client application to the target is first buffered in the RTDX Host Library. The data remains in the RTDX Host Library until a request for data arrives from the target. Once the RTDX Host Library has enough data to satisfy the request, it writes the data to the target in such a way as to not interfere with the target application.

The state of the buffer is returned into the variable bufferstate. A positive value indicates the number of bytes the RTDX Host Library has buffered, which the target has not yet requested. A negative value indicates the number of bytes that the target has requested, which the RTDX Host Library has not yet satisfied.

The purpose of this lesson is to learn how to write a RTDX host client that can send a single integer to the target application. The example used in this lesson shows how to send integer value 5 to the target.

Over the course of the lesson, you will learn about the following types of files:

- .c This file contains source code that provides the main functionality of this project
- .h This file declares the buffer C-structure as well as define any required constants

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007

- .plt This file contains all of your project build and configuration options
- .bas This file uses Visual Basic or Excel to create code changes to the project.

To send data to the target, you must follow this procedure:

- Open your application code in either a Visual Basic or an Excel project.
- Insert specific RTDX syntax in your application code to allow real-time data exchange.
- Process the data and test your application code.

Before you begin, [disable RTDX](#).

1. Open the example application code, `SZL4.bas`, in either a Visual Basic or an Excel project.

The path for the example source file, `SZL4.bas`, is `C:\CCStudio_v3.1.0\tutorial\target\sect_2\less_4`. Import the source file into either a Visual Basic or an Excel project and view the application code. For information on importing files, see the Microsoft Visual Basic or the Excel Help.

2. Send (write) data to the target after the data = VALUE\_TO\_SEND instruction. The example in this lesson sends a 32-bit integer to the target. Please see the Example Visual

```
status = rtdx.write(4,data, bufferstate)
```

[Example Visual Basic Code SZL4.bas](#)

Click [here](#) to look at the Visual C++ version of the Visual Basic code `SZL4.bas`.

To process the data and test your application code, follow these steps:

1. If you have not already done so, from the Windows Start menu, choose Programs→Texas Instruments→Code Composer Studio 3.1→Code Composer Studio 3.1. (Or, double-click the Code Composer Studio icon on your desktop.)
2. From the Project menu in Code Composer, select Open.
3. Browse to the `C:\CCStudio_v3.1.0\tutorial\target\sect_2\less_4` directory, select `SZL4.plt` and click Open. If you configured your target for big endian, select `SZL4_e.plt`.
4. From the Project menu, select Build to build your example target application executable, `SZL4.out`.
5. From the File menu, select Load Program.
6. In the Load Program dialog box, browse to the `C:\CCStudio_v3.1.0\tutorial\target\sect_2\less_4\Debug` directory and select `SZL4.out`.

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007

Getting Started With the Code Composer Studio Tutorial

Page 460 sur 548

7. Click Open.
8. In Code Composer, select Tools→RTDX→Configuration Control.
9. In the Configuration Control window, click inside the Enable RTDX checkbox to enable RTDX.
10. From the Debug menu, select Run to run your application.

11. Run the Visual Basic code , `SZL4.bas`, in either Visual Basic or Excel. For updated information on running code, see the Microsoft Visual Basic or the Excel Help.

The message, "Value 5 was received from the host," appears in the Code Composer Standard Output (Stdout) window, verifying that the host client has sent integer value 5 to the target.

Before you move to the next lesson, [disable RTDX](#), and close the `SZL4` project and all open windows in CCS.



## Remote Enabling and Disabling of a Channel

RTDX - L3

RTDX allows you to remotely enable and disable channels on the target from the RTDX host client application. This allows the client application to alter or control the data that the target sends without altering the target application.

The purpose of this lesson is to learn how to remotely enable and disable channels from the host client. The example used in this lesson shows how to enable the target channel ochan, receive integer value 5 from the target, and disable the channel.

To remotely enable or disable a channel, you must follow this procedure:

- Open your application code in either a Visual Basic or an Excel project.
- Insert specific RTDX syntax in your application code to allow remote enabling and disabling of a channel.
- Process the data and test your application code.

Before you begin, [disable RTDX](#).

1. Open the example application code, `SZL9.bas`, in either a Visual Basic or an Excel project.

The path for the example source file, `SZL9.bas`, is `C:\CCStudio_v3.1.0\tutorial\target\sect_2\less_9`. Import the source file into either a Visual Basic or an Excel project and view the application code. For information on importing files, see the Microsoft Visual Basic or the Excel Help.

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007

2. First, you must enable a channel for target writes. Enabling calls in the target application can override remote enabling of a channel from the host. The example in this lesson uses the name `ochan` to define an output channel. Insert this code after the `Set rdx = CreatedObject("RTDX")` code in the `SZL9.bas` file. Please see the Example Visual Basic Code `SZL9.bas` [link](#) to view the modified file.
 

```
status = rtdx.EnableChannel(CHANNEL_NAME)
```

3. Then you must disable the channel from target writes. Disabling calls in the target application can override remote disabling of a channel from the host. Insert this code before the `status = rtdx.Close()` instruction. Please see the Example Visual Basic Code `SZL9.bas` [link](#) to view the modified file.
 

```
status = rtdx.DisableChannel(CHANNEL_NAME)
```

#### [Example Visual Basic Code SZL9.bas](#)

**Note:** Before you proceed further, make sure that the Visual Basic Immediate window is visible.

To process the data and test your application code, follow these steps:

1. From the Project menu in Code Composer, select Open.
2. Browse to the `C:\CCStudio_v3.10\tutorial\target\sect_2\less_9` directory, select `SZL9.pjt` and click Open. If you configured your target for big endian, select `SZL9_e.pjt`.
3. From the Project menu, select Build to build your example target application executable, `SZL9.out`.
4. From the File menu, select Load Program.
5. In the Load Program dialog box, browse to the `v\tutorial\target\sect_2\less_9\Debug` directory and select `SZL9.out`.
6. Click Open.
7. In Code Composer, select Tools→RTDX→Configuration Control.
8. In the Configuration Control window, click inside the Enable RTDX checkbox to enable RTDX.
9. From the Debug menu, select Run to run your application.

10. Run the Visual Basic code , `SZL9.bas`, in either Visual Basic or Excel. For updated information on running code, see the Microsoft Visual Basic or the Excel Help.

The message, "Value 5 was received from the target," appears in the Visual Basic Immediate window, verifying that the host client has enabled a channel for a read.

Before you move to the next lesson, [disable RTDX](#) , and close the `SZL9` project and all open windows in CCS.

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007

## Getting Started With the Code Composer Studio Tutorial

Page 462 sur 548

This concludes the Writing RTDX Host Clients lesson. For further tutorials on writing RTDX host clients, please click on the In-Depth Topics book in the Table of Contents menu and choose a lesson, or follow these links:

[Receiving Entire Messages Using SAFEARRAYS](#)

[Sending an Array of Integers Using SAFEARRAYS](#)

[Receiving Data from Multiple Channels](#)

[Receiving Data from the ALL Channel](#)

[Seeking Through Messages](#)



## [Receiving Entire Messages Using SAFEARRAYS](#)

**RTDX - L3**

RTDX allows you to obtain entire messages sent from the target through the use of SAFEARRAYS. SAFEARRAYS are COM structures that identify the type and quantity of data contained in the array. Using SAFEARRAYS to receive RTDX data provides greater efficiency by retrieving the entire message from the target.

The purpose of this lesson is to learn how to use the host functions `ReadSA*` to write a RTDX host client that can read entire messages (arrays of data) from the target. The example used in this lesson shows how to read integer values 1 to 10 from the target.

To receive data from the target, you must follow this procedure:

- Open your application code in either a Visual Basic or an Excel project.
- Insert specific RTDX syntax in your application code to allow real-time data exchange.
- Process the data and test your application code.

Before you begin, [disable RTDX](#) and [open](#) the example application code, `SZL3.bas`, in either a Visual Basic or an Excel project.

1. First, declare a variable of type Variant. This variable contains a pointer to a SAFEARRAY. Insert the following code after the `Dim rdx As Object` instruction. Please see the Example Visual Basic Code `SZL3.bas` [link](#) to view the modified file.
 

```
Dim vardata As Variant
```
2. Then, read entire messages using SAFEARRAYS by inserting this code before the `Select Case (status)` instruction. Please see the Example Visual Basic Code `SZL3.bas` [link](#) to view the modified file.

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007

```
status = rtdx.ReadSAL4(vardata)
```

3. Finally, print each element in the array by inserting this code after the Case Success instruction. Please see the Example Visual BasicCode S2L3.bas [link](#) to view the modified file.

```
For i = Lbound(vardata) To Ubound(vardata)
Debug-Print vardata(i)
Next i
```

#### [Example Visual Basic Code S2L3.bas](#)

Click [here](#) to look at the Visual C++ version of the Visual Basic code S2L3.bas.

**Note:** Before you proceed further, make sure that the Visual Basic Immediate window is visible.

To process the data and test your application code, follow these steps:

- [Save and build your target application to produce the target application executable. \\*.OUT.](#)
- [Load your target application executable.](#)
- [Enable RTDX and run your target application.](#)
- [Run the host client.](#)

Numbers ranging from 1 to 10 appear in the Visual Basic Immediate window, verifying that the host client has read integer values 1 to 10 from the target.

Before you move to the next lesson, [disable RTDX](#) , and close the S2L3 project and all open windows in CCS.

[Note on COM Error](#)



## **Sending an Array of Integers Using SAFEFARRAYS**

**RTDX - L3**

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007

## Getting Started With the Code Composer Studio Tutorial

Page 464 sur 548

Using SAFEFARRAYs to send RTDX data provides greater efficiency by sending the entire message from the host to the target. The purpose of this lesson is to learn how to use the Write function to write a RTDX host client that can send entire messages (arrays of data) to the target. The example used in this lesson shows how to send integer values 1 to 10 to the target.

To send data to the target, you must follow this procedure:

- Open your application code in either a Visual Basic or an Excel project.
- Insert specific RTDX syntax in your application code to allow real-time data exchange.
- Process the data and test your application code.

Before you begin, [disable RTDX](#) and [open](#) the example application code, S2L5.bas, in either a Visual Basic or an Excel project.

1. First, create and initialize a SAFEFARRAY of data to send to the target. Insert the following code after the first End If instruction in the bas file. Please see the Example Visual Basic Code S2L5.bas [link](#) to view the modified file.

```
data = 1
For i = Lbound(arraydata) To Ubound(arraydata) - 1)
arraydata(i) = data
data = data + 1
Next i
```

2. Finally, send (write) data to the target by inserting this code after the previous code. The example in this lesson writes a SAFEFARRAY of 32-bit integers to the target. Please see the Example Visual Basic Code S2L5.bas [link](#) to view the modified file.

```
status = rtdx.Writer(Var(arraydata), bufferSize)
```

#### [Example Visual Basic Code S2L5.bas](#)

Click [here](#) to look at the Visual C++ version of the Visual Basic code S2L5.bas.

To process the data and test your application code, follow these steps:

- [Save and build your target application to produce the target application executable. \\*.OUT.](#)
- [Load your target application executable.](#)
- [Enable RTDX and run your target application.](#)
- [Run the host client.](#)

Ten lines of output with numbers ranging from 1 to 10 appear in the Code Composer Standard Output (Stdout) window, verifying that the host client has sent integer values 1 to 10 to the target.

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007



Before you move to the next lesson, [disable RTDX](#) , and close the SZL5 project and all open windows in CCS.



## Receiving Data From Multiple Channels

### RTDX - L3

RTDX supports host clients receiving data from multiple channels simultaneously. The data available from a channel is independent of the activity or data available from other channels. Hence, reading data from one channel does not affect the data read from other channels.

The purpose of this lesson is to learn how to receive target-sent data from separate channels. The example used in this lesson shows how to receive integer value 3 from channel ochan1, integer value 2 from channel ochan2, and integer value 1 from channel ochan3.

To receive target-sent data from separate channels, you must follow this procedure:

- Open your application code in either a Visual Basic or an Excel project.
- Insert specific RTDX syntax in your application code to allow real-time data exchange.
- Process the data and test your application code.

Before you begin, [disable RTDX](#) and [open](#) the example application code ,SZL6.bas, in either a Visual Basic or an Excel project.

1. Create and initialize an array of channel attributes, such as name and type. This allows you to conveniently iterate through the array to open the channels. Insert this code after the 'The end of the log file has been read/' instruction. Please see the Example Visual Basic Code SZL6.bas [link](#) to view the modified file.

```
Public Type RTDX_Channel
    obj As Object
    name As String
    type As String
    opened As Boolean
End Type
```

2. Create an array instance of the type RTDX\_Channel. Insert this code after the Sub main() instruction. Please see the Example Visual Basic Code SZL6.bas [link](#) to view the modified file.
 

```
Dim rtdx(1 To 3) As RTDX_Channel
```
3. Initialize the channel name. The example in this lesson uses the arbitrary channel names ochan1, ochan2, and ochan3 to define multiple output channels. Insert this code after the On Error Goto ErrorHandler instruction. Please see the Example Visual Basic Code SZL6.bas [link](#) to view the modified file.
 

```
rtdx(1).name = "ochan1"
rtdx(2).name = "ochan2"
```

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm 26/09/2007

## Getting Started With the Code Composer Studio Tutorial

4. After the previous instructions, create an instance of the RTDX internal COM object, obtain a pointer to the RTDX COM object, and open the channels. Please see the Example Visual Basic Code SZL6.bas [link](#) to view the modified file.

```
For i = LBound(rtdx) To UBound(rtdx)
    rtdx(i).type = "R"
    rtdx(i).opened = False
    Set rtdx(i).obj = CreateObject("RTDX")
    status = rtdx(i).obj.Open(rtdx(i).name, _
        rtdx(i).type)
    If status <> Success Then
        Debug.Print "Opening of channel " & _
            rtdx(i).name & _
            " failed" & _
            Chr(10)
    Else
        rtdx(i).opened = True
    End If
Next i
```

5. Read the data from the channel. The example in this lesson reads a 32-bit integer from the target. Insert the following code after the If rtdx(i).opened Then instruction. Please see the Example Visual Basic Code SZL6.bas [link](#) to view the modified file.
 

```
status = rtdx(i).obj.ReadI(data)
```
6. Close the channel from reading and release the reference to the RTDX COM object. Insert this code before the Exit Sub instruction. Please see the Example Visual Basic Code SZL6.bas [link](#) to view the modified file.

```
For i = LBound(rtdx) To UBound(rtdx)
    status = rtdx(i).obj.Close()
    Set rtdx(i).obj = Nothing
Next i
```

[Example Visual Basic Code SZL6.bas](#)

Note: Before you proceed further, make sure that the Visual Basic Immediate window is visible.

To process the data and test your application code, follow these steps:

- [Save and build your target application to produce the target application executable. \\*.OUT.](#)
- [Load your target application executable.](#)
- [Enable RTDX and run your target application.](#)

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

- [Run the host client.](#)

The following three messages appear in the Visual Basic Immediate window:

```
Value 3 was received from target via channel ochan1  
Value 2 was received from target via channel ochan2  
Value 1 was received from target via channel ochan3
```

This verifies that the host client has read integer values 1, 2, and 3 from the specified channels.

Before you move to the next lesson, [disable RTDX](#), and close the S2L6 project and all open windows in CCS.



---

### Receiving Data From the ALL Channel

**RTDX - L3**

Messages are received by the host in the order that they are sent from the target. However, the order in which data is received from multiple channels does not necessarily reflect the order that the data was written with respect to each channel. The ALL channel is a special channel, which allows the host client to receive all the data in the order it was sent by the target application, regardless of the channel to which it was written. This provides an easy mechanism to receive messages on the host in a sequential order, from a single channel.

The purpose of this lesson is to learn how to receive data sent to the host using the ALL channel. The example used in this lesson shows how integer values 1, 2, and 3 are received from the target in a sequential order, via the ALL channel.

To receive data from the target using the ALL channel, you must follow this procedure:

- Open your application code in either a Visual Basic or an Excel project.
- Insert specific RTDX syntax in your application code to allow real-time data exchange.
- Process the data and test your application code.

Before you begin, [disable RTDX](#) and [open](#) the example application code, S2L7.bas, in either a Visual Basic or an Excel project.

- Open the ALL channel to receive data by inserting this code after the Set rtdx = CreateObject("RTDX") instruction. Please see the [Example Visual Basic Code S2L7.bas link](#) to view the modified file.

```
status = rtdx.Open("ALL", "R")
```

**Note:** Do not write to the ALL channel, as this can have unpredictable results.

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007

Getting Started With the Code Composer Studio Tutorial

Page 468 sur 548

[Example Visual Basic Code S2L7.bas](#)

**Note:** Before you proceed further, make sure that the Visual Basic Immediate window is visible.

To process the data and test your application code, follow these steps:

- [Save and build your target application to produce the target application executable. \\*.OUT.](#)
- [Load your target application executable.](#)
- [Enable RTDX and run your target application.](#)
- [Run the host client.](#)

The following three messages appear in the Visual Basic Immediate window:

```
Value 1 was received from the target  
Value 2 was received from the target  
Value 3 was received from the target
```

This verifies that the host client has read integer values 1, 2, and 3 in the order that they were sent by the target.

Before you move to the next lesson, [disable RTDX](#), and close the S2L7 project and all open windows in CCS.



---

### Seeking Through Messages

**RTDX - L3**

RTDX allows you to move forward or backward through a log of messages using the Seek and Rewind functions. The purpose of this lesson is to learn how to seek through a log of messages to read a specific value. The example in this lesson shows how to receive integer values 1 to 10 from the target, then use the Seek() function to read these values in reverse order.

To seek through messages, you must follow this procedure:

- Open your application code in either a Visual Basic or an Excel project.
- Insert specific RTDX syntax in your application code to allow real-time data exchange for "seeking" messages.

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007

- Process the data and test your application code.

Before you begin, [disable RTDX](#) and [open](#) the example application code, S2L8.bas, in either a Visual Basic or an Excel project.

1. Get the total number of messages received by the host. Insert this code before the first If status <> Success Then instruction. Please see the Example Target Application C Code S2L8.bas [link](#) to view the modified file.

```
status = rtdx.GetNumMsgs(nummsgs)
```

2. Rewind the log of data. Insert this code after the first Next i instruction. Please see the Example Visual Basic Code S2L8.bas [link](#) to view the modified file.

```
status = rtdx.Rewind()
```

3. Seek to a specific message in the log of data. Insert this code after the For i = nummsgs To 1 Step -1 instruction. Please see the Example Visual Basic Code S2L8.bas [link](#) to view the modified file.

```
status = rtdx.Seek(i)
```

#### [Example Visual Basic Code S2L8.bas](#)

**Note:** Before you proceed further, make sure that the Visual Basic Immediate window is visible.

To process the data and test your application code, follow these steps:

- [Save and build your target application to produce the target application executable, \\*.OUT.](#)
- [Load your target application executable.](#)
- [Enable RTDX and run your target application.](#)
- [Run the host client.](#)

The following messages appear in the Visual Basic Immediate window:

```
Reading data from beginning to end
Value 1 was received from the target
Value 2 was received from the target
Value 3 was received from the target
Value 4 was received from the target
Value 5 was received from the target
Value 6 was received from the target
Value 7 was received from the target
```

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007

Getting Started With the Code Composer Studio Tutorial

Page 470 sur 548

```
Value 8 was received from the target
Value 9 was received from the target
Value 10 was received from the target
```

```
Reading data from end to beginning
Value 10 was received from the target
Value 9 was received from the target
Value 8 was received from the target
Value 7 was received from the target
Value 6 was received from the target
Value 5 was received from the target
Value 4 was received from the target
Value 3 was received from the target
Value 2 was received from the target
Value 1 was received from the target
```

This verifies that the host client has received integer values 1 to 10 from the target, and read these values in reverse order.

Before you move to the next lesson, [disable RTDX](#), and close the S2L8 project and all open windows in CCS.



## Overview

RTDX - L4

RTDX has several functionalities that can be used to perform complicated tasks and improve RTDX performance. This lesson discusses a few of these functionalities.

### Learning Objectives:

- Flush a Channel
- Send Structured Data
- Communicate With a Processor in a Multiprocessor Environment

### Example: sect\_4

#### Application Objective:

The sect\_4 examples are used only in Lessons S4L2, S4L3, and S4L6. In these lessons, you learn what code to insert into the sect\_4 examples to build a target application that can perform complex RTDX tasks. The remaining lessons provide you with information to improve RTDX performance. The final code documents for each example in this lesson are available for review in the Example Codes Reference Library in the Table of Contents under RTDX.

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007

This lesson consists of the following steps:

[Flushing a Channel](#)

[Sending Structured Data](#)

[Communicating With a Processor in a Multiprocessor Environment](#)



The forward arrow will take you to the next page in this lesson.

## Flushing a Channel

---

### RTDX - L4

When the target application requests to read a certain amount of data, but the host client wishes to send less data, the host client can call the flush() routine to send less data to the target. The read request is then considered satisfied. The flush() routine is part of the host COM API.

The purpose of this lesson is to understand how to successfully flush a channel, taking into account certain considerations relating to the flush() routine.

When using the flush() routine, you must consider the following:

- The flush() routine is meant for channels opened by a host client for writing (target application reading).
- A host client must be careful of which read request to flush. If a target application posts a read request and the host client satisfies the request with a write and then flushes the channel, there is a possibility that the host client may flush a subsequent target read request. Use the function, StatusQWrite(), to check how much data the target application needs.

The example used in this lesson shows how to send 2 bytes of data to the target and then use the flush() routine to flush a channel.

Over the course of the lesson, you will learn about the following types of files:

- .c This file contains source code that provides the main functionality of this project
- .h This file declares the buffer C-structure as well as define any required constants
- .jdt This file contains all of your project build and configuration options
- .bas This file uses Visual Basic or Excel to create code changes to the project.

To flush a channel when sending data to the target, you must follow this procedure:

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007

## Getting Started With the Code Composer Studio Tutorial

Page 472 sur 548

- Open your application code in either a Visual Basic or an Excel project.
- Insert specific RTDX syntax in your application code to allow real-time data transfer and flush a channel.
- Process the data and test your application code.

Before you begin, [disable RTDX](#).

1. Open the example application code, S4L2.bas, in either a Visual Basic or an Excel project.

The path for the example source file, S4L2.bas, is C:\CCStudio\_v3.10\tutorial\target\sect\_4\less\_2. Import the source file into either a Visual Basic or an Excel project and view the application code. For information on importing files, see the Microsoft Visual Basic or the Excel Help.

2. First, you must verify any data requests from the host to find out if the target application has requested data from the host. Insert the following code into S4L2.bas after the status = rtdx.WritedId(data, bufferSize) instruction. Please see the Example Visual Basic Code S4L2.bas [link](#) to view the modified file.

```
status = rtdx.StatusQWrite(bufferstate)
```
3. Finally, you should flush the channel. Insert the following code after the If bufferstate <= 0 Then instruction. Please see the Example Visual Basic Code S4L2.bas [link](#) to view the modified file.

```
status = rtdx.Flush()
```

### Example Visual Basic Code S4L2.bas

To process the data and test your application code, follow these steps:

1. From the Project menu in Code Composer, select Open.
2. Browse to the C:\CCStudio\_v3.10\tutorial\target\sect\_4\less\_2 directory, select S4L2.jdt and click Open. If you configured your target for big endian, select S4L2\_e.jdt.
3. From the Project menu, select Build to build your example target application executable, S4L2.out.
4. From the File menu, select Load Program.
5. In the Load Program dialog box, browse to the C:\CCStudio\_v3.10\tutorial\target\sect\_4\less\_2\Debug directory and select S4L2.out.
6. Click Open.
7. In Code Composer, select Tools-->RTDX-->Configuration Control.
8. In the Configuration Control window, click inside the Enable RTDX checkbox to enable RTDX.

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007

9. From the Debug menu, select Run to run your application.

10. [Run the Visual Basic code](#) , S4L2.bas, in either Visual Basic or Excel. For updated information on running code, see the Microsoft Visual Basic or the Excel Help.

The following message appears in the Code Composer Standard Output (Stdout) window:

"Target application requested 2 bytes and only received 1 byte."

This verifies that the host client sent 2 bytes to the target and then successfully flushed the channel. Before you move to the next lesson, [disable RTDX](#) , and close the S4L2 project and all open windows in CCS.



## Sending Structured Data

### RTDX - L4

It is possible to send data to the host in the form of a structure. This means that data of various lengths can be sent to the host in groups. The example used in this lesson shows how to send a structure of three integers, with values 1, 2, and 3 to the host.

To send data to the host, you must follow this basic procedure:

- Open your project and view your application code in Code Composer.
- Insert specific RTDX syntax in your application code to allow real-time data transfer.
- Process the data and test your application code.

To open your project and modify the application code, follow these steps:

1. If you have not already done so, from the Windows Start menu, choose Programs→Texas Instruments→Code Composer Studio 3.1→Code Composer Studio 3.1. (Or, double-click the Code Composer Studio icon on your desktop.)
2. From the Project menu, select Open.
3. Browse to the `C:\CCStudio_v3.10\tutorial\target\sect_4\less_3` directory, and select S4L3.plt and click Open. If you configured your target for big endian, select S4L3\_e.plt.
4. If the Project View window is not open, click on View→Project.
5. In the project view, expand the Project list by clicking the + signs next to Projects, S4L3.plt, Source, and open the S4L3.c file.
6. In the S4L3.c file, you must create a structure to be sent to the host. The example in this lesson declares a structure of three integers, with values 1, 2, and 3. Insert the following

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007

## Getting Started With the Code Composer Studio Tutorial

Page 474 sur 548

code after the #define VALUE\_FOR\_SDATA3 3 instruction. Please see the Example Target Application C Code S4L3.c [link](#) to view the modified file.

```
struct {
  int s_data1;
  int s_data2;
  int s_data3;
}mystruct;
```

7. Finally, send the structured data to the host by inserting the following code after the `RTDX_enableOutput(&ochan );` instruction. Please see the Example Target Application C Code S4L3.c [link](#) to view the modified file.

```
status = RTDX_write( &ochan, &mystruct, sizeof(mystruct) );
```

[Example Target Application C Code S4L3.c](#)

To process the data and test your application code, follow these steps:

1. From the File menu, select Save to save your example source file, S4L3.c.
2. From the Project menu, select Build to build your example target application executable, S4L3.out.

**Note:** Be aware of compiler options that may affect RTDX performance. For example, the C6x compiler interrupt threshold option, `-mi<n>`, can be used to limit the length of interrupt lockouts in compiled code. For more information, see the TMS320C6000 Optimizing Compiler User's Guide (SPRU187).

3. From the File menu, select Load Program.
4. In the Load Program dialog box, browse to the `C:\CCStudio_v3.10\tutorial\target\sect_4\less_3\Debug` directory and select S4L3.out.
5. Click Open.
6. In the main menu, select Tools→RTDX→Configuration Control.
7. In the Configuration Control window, click inside the Enable RTDX checkbox to enable RTDX.
8. From the Debug menu, select Run to run your application.
9. Open an MS-DOS™ Prompt window and type `C:\CCStudio_v3.10\tutorial\target\sect_4\less_3\S4L3.exe` to invoke the example executable host client.

The numbers 1, 2, and 3 appear in the MS-DOS Prompt window, verifying that the host client has read integer values 1, 2, and 3 sent by the target application.



file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007

## Communicating with a Processor in a Multiprocessor Environment

RTDX - L4

You can use the SetProcessor() routine to establish communication with a specific processor in a multiprocessor environment. A multiprocessor environment is an environment in which more than one processor is configured under the Code Composer Setup. The SetProcessor() routine is part of the host COM API. The purpose of this lesson is to understand how to successfully communicate with a processor in a multiprocessor environment, taking into account certain considerations relating to the SetProcessor() routine.

When using the SetProcessor() routine, you must consider the following:

- The SetProcessor() routine must be called immediately after you obtain a reference to the RTDX COM object.
- A call to the SetProcessor() routine fails if it is called after any other method is called.

To communicate with a processor in a multiprocessor environment, you must follow this basic procedure:

- Open your application code in either a Visual Basic or an Excel project.
- Insert specific RTDX syntax in your application code to establish communication with a processor in a multiprocessor environment.
- Process the data and test your application code.

Before you begin, [disable RTDX](#).

1. Open the example application code, S4L6.bas, in either a Visual Basic or an Excel project.

The path for the example source file, S4L6.bas, is `C:\CCStudio_v3.10\tutorial\target\sect_4\less_6`. Import the source file into either a Visual Basic or an Excel project and view the application code. For information on importing files, see the Microsoft Visual Basic or the Excel Help.

2. **Specify the board and processor** . Insert this code after the `Set rtdx = CreateObject("RTDX")` instruction. Please see the Example Visual Basic Code S4L6.bas [link](#) to view the modified file. Please use the board and CPU that are appropriate for your target.

```
board = "G6xxx EVM (Texas Instruments)"  
cpu = "TMS320C6701EVM"
```

3. Attach to the specified board and processor by specifying the following code after the previous instructions. Please see the Example Visual Basic Code S4L6.bas [link](#) to view the modified file.

```
status = rtdx.SetProcessor(board, cpu)
```

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007

## Getting Started With the Code Composer Studio Tutorial

Page 476 sur 548

[Example Visual Basic Code S4L6.bas](#)

Click [here](#) to look at the Visual C++ version of the Visual Basic code S4L6.bas.

**Note:** Before you proceed further, make sure that the Visual Basic Immediate window is visible.

To process the data and test your application code, follow these steps:

1. From the Project menu in Code Composer, select Open.
2. Browse to the `C:\CCStudio_v3.10\tutorial\target\sect_4\less_6` directory, select S4L6.pjt and click Open. If you configured your target for big endian, select S4L6\_e.pjt.
3. From the Project menu, select Build to build your example target application executable, S4L6.out.
4. From the File menu, select Load Program.
5. In the Load Program dialog box, browse to the `C:\CCStudio_v3.10\tutorial\target\sect_4\less_6\Debug` directory and select S4L6.out.
6. Click Open.
7. In Code Composer, select Tools→RTDX→Configuration Control.
8. In the Configuration Control window, click inside the Enable RTDX checkbox to enable RTDX.
9. From the Debug menu, select Run to run your application.
10. **Run the Visual Basic code** , S4L6.bas, in either Visual Basic or Excel. For updated information on running code, see the Microsoft Visual Basic or the Excel Help.

The message, "S" appears in the Visual Basic Immediate window, verifying that the host client has received integer value 5 from the target.

This concludes the Advanced RTDX lesson.



## Target Application C Code S1L1.c

RTDX-S1

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007

The following is the finished C code *SLLI.c*.

#### Complete Code

```

/*****
 * SLLI.c - (Section 1, Lesson 1) SENDING AN INTEGER TO THE HOST
 * This is the RTDX Target Code for Section 1, Lesson 1
 *
 * This example sends integer value 5 to the host
 *****/

#include <rtdx.h>,
 * which defines the RTDX target API.
/*****
#include <rtdx.h>
*****/

This example includes <target.h>, which defines TARGET_INITIALIZE.
*****/
#include "target.h" /* defines TARGET_INITIALIZE() */
#include <stdio.h> /* C_I/O */
/* Define the value to send to the host */
#define VALUE_TO_SEND 5
*****/

Declare a global output channel;
 * the parameter name in this example must be "ochan."

```

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007

Getting Started With the Code Composer Studio Tutorial

Page 478 sur 548

```

*****/
RTDX_CreateOutputChannel( ochan );

void main()
{
/*****
 * Declare variables for value to send to host and to check status.
 *****/
int data = VALUE_TO_SEND;
int status;
*****/

Initialize target using the macro
 * TARGET_INITIALIZE or insert your own code here.
*****/

Enable the output channel.
*****/
RTDX_enableOutput( ochan );
*****/

Send data to the host.
/*****
status = RTDX_write( ochan, &data, sizeof(data) );

```

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007

```

/*****
 * Check return status of RTDX_write().
 *****/
if ( status == 0 ) {
    puts( "ERROR: RTDX_write failed!\n" );
    exit( -1 );
}
while ( RTDX_writing != NULL ) {
    #if RTDX_POLLING_IMPLEMENTATION
    RTDX_Poll();
    #endif
}
*****
Disable the output channel.
*****
RTDX_disableOutput ( kochan );
puts( "Program Complete!\n" );
}

```

### Target Application C Code S1L2.c

RTDX-S1

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007

### Getting Started With the Code Composer Studio Tutorial

Page 480 sur 548

The following is the finished C code S1L2.c.

#### Complete Code

```

/*****
 * S1L2.c - (Section 1, Lesson 2) SENDING AN ARRAY OF INTEGERS TO
 * THE HOST
 * This is the RTDX Target Code for Section 1, Lesson 2
 *
 * This example sends integer values 1-10 to the host
 *****/
#include <rtdx.h> /* defines RTDX target API calls */
#include "target.h" /* defines TARGET_INITIALIZE() */
#include <stdio.h> /* C I/O */

/* These are the values we are going to send to the host */
#define VALUES_TO_SEND {1,2,3,4,5,6,7,8,9,10}

/* declare a global output channel */
RTDX_CreateOutputChannel( ochan );

void main()
{
    int arraydata[] = VALUES_TO_SEND;
    int status;
    TARGET_INITIALIZE();
    /* enable the output channel */
    RTDX_enableOutput( kochan );
}

```

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007



```

*****
Send data to the host:
* the parameter name in this example must be "ochan,"
*****
status = RTDX_write( &ochan, arraydata, sizeof(arraydata) );

if ( status == 0 ) {
    puts( "ERROR: RTDX_write failed!\n" );
    exit( -1 );
}

while ( RTDX_write != NULL ) {

    #if RTDX_POLLING_IMPLEMENTATION
    RTDX_Poll();
    #endif

    #endif
}

/* disable the output channel */
RTDX_disableOutput( &ochan );
puts( "Program Complete!\n" );
}

```

### Target Application C Code S1L3.c

RTDX-S1

The following is the finished C code S1L3.c.

Complete Code

```

/*****
* S1L3.c - (Section 1, Lesson 3) RECEIVING AN INTEGER FROM
* THE HOST
* This is the RTDX Target Code for Section 1, Lesson 3
* This example receives integer value 5 from the host
*****

```

```

#include <rtdx.h> /* defines RTDX target API calls */
#include "target.h" /* defines TARGET_INITIALIZE() */
#include <stdio.h> /* C_I/O */
*****
Create a global input channel;
* the parameter name in this example must be "ichan."
*****
RTDX_CreateInputChannel( ichan );

void main()
{
    int data;
    int status;
    TARGET_INITIALIZE();
    *****
    Enable the input channel.
    *****
    RTDX_enableInput( &ichan );
    *****
    Request and wait to receive an integer.
    *****
    status = RTDX_read( &ichan, &data, sizeof(data) );
    if ( status != sizeof(data) ) {
        printf( "ERROR: RTDX_read failed!\n" );
        exit( -1 );
    } else

```

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007

Getting Started With the Code Composer Studio Tutorial

Page 482 sur 548

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007

```

printf( "Value %d was received from the host\n", data );
*****
Disable the input channel.
*****
RTDX_disableInput( &ichan );
printf( "Program Complete!\n" );
}

```

### Target Application C Code S1L4.c

RTDX-S1

The following is the finished C Code S1L4.c

#### Complete Code

```

/*****
* S1L4.c - (Section 1, Lesson 4) RECEIVING AN INTEGER FROM THE *
* HOST ASYNCHRONOUSLY *
* This is the RTDX Target Code for Section 1, Lesson 4 *
* This example receives integer value 5 from the host *
*****
#include <rtdx.h> /* defines RTDX target API calls */
#include "target.h" /* defines TARGET_INITIALIZE() */
#include <stdio.h> /* C_I/O */

/* declare a global input channel */
RTDX_CreateInputChannel( &ichan );

```

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhB A53.htm

26/09/2007

Getting Started With the Code Composer Studio Tutorial

Page 484 sur 548

```

void main()
{
    int data;
    int status;

    TARGET_INITIALIZE();

    /* enable the input channel */
    RTDX_enableInput( &ichan );

    *****
    *Request an integer from the host;
    * the parameter name in this example must be "ichan."
    *****
    status = RTDX_readB( &ichan, &data, sizeof(data) );

    if ( status != RTDX_OK ) {
        printf( "ERROR: RTDX_readB failed!\n" );
        exit( -1 );
    }

    status = 1;
    while ( status ) {
        *****
        * Loop until channel is no longer busy waiting to receive data.
        *****
        status = RTDX_channelBusy( &ichan );

        /*
        * useful processing can be done here while waiting
        * for data
        */

        #if RTDX_POLLING_IMPLEMENTATION

        /*
        * if RTDX uses polling on this target, then be sure
        * to call it regularly to transfer the data
        */
        RTDX_Poll();

        #endif
    }

    *****
    * Verify that your program has received all the requested data.

```

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhB A53.htm

26/09/2007

```

*****
status = RTDX_sizeofInput( ichan );

if ( status != sizeof(data) )
    printf( "ERROR: Did not receive all the data!\n" );
else
    printf( "Value %d was received from the host!\n", data );

/* disable the input channel */
RTDX_disableInput( ichan );
printf( "Program Complete!\n" );
}

```

### Target Application C Code S1L5.c

RTDX-S1

The following is the finished C code S1L5.c.

#### Complete Code

```

/*****
* S1L5.c - (Section 1, Lesson 5) RECEIVING AN ARRAY OF INTEGERS
* FROM THE HOST
* This is the RTDX Target Code for Section 1, Lesson 5
* This example receives integer values 1-10 from the host
*****
#include <rtdx.h> /* defines RTDX target API calls */
#include "target.h" /* defines TARGET_INITIALIZE() */
#include <stdio.h> /* C_I/O */

/* declare a global input channel */
RTDX_CreateInputChannel( ichan );

```

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhB A53.htm

26/09/2007

### Getting Started With the Code Composer Studio Tutorial

Page 486 sur 548

```

void main()
{
    int arraydata[10];
    int status, i;

    TARGET_INITIALIZE();

    /* enable the input channel */
    RTDX_enableInput( ichan );
    *****
    * Request an entire array of integers;
    *****
    * the parameter name in this example must be "ichan."
    *****
    status = RTDX_read( &ichan, arraydata, sizeof(arraydata) );
    if ( status != sizeof(arraydata) ) {
        printf( "ERROR: RTDX_read failed!\n" );
        exit( -1 );
    } else
        for( i = 0; i < ( sizeof(arraydata) / sizeof(int) ); i++)
            printf( "Value %d was received from the host!\n",
                arraydata[i] );

    /* disable the input channel */
    RTDX_disableInput( ichan );
    printf( "Program Complete!\n" );
}

```

### Visual Basic Code S2L2.bas

RTDX-S2

The following is the finished Visual Basic code S2L2.bas.

#### Complete Code

#### Option Explicit

```

*****

```

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhB A53.htm

26/09/2007

```

** S212.bas - Section 2, Lesson 2) RECEIVING AN INTEGER FROM THE
* TARGET
** This is the RTDX Host Client for Section 2, Lesson 2
**
** This example receives integer value 5 from the target
*****
' Include the RTDX return code constants.
*****
Const Success = #H0 ' Method call is valid
Const Failure = #H80004005 ' Method call failed

Const ENODataAvailable = #H8003001E ' No data was available.
' However, more data may be
' available in the future.

Const ENODflogFile = #H80030002 ' No data was available.
' The end of the log file has
' been reached.

Sub main()
*****
' Declare a variable of type Object. This will be used to
' access the methods of the interface.
*****
Dim rtdx As Object

Dim data As Long
Dim status As Long

On Error GoTo Error_Handler

```

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007

Getting Started With the Code Composer Studio Tutorial

Page 488 sur 548

```

*****
' Create an instance of the RTDX COM Object.
*****
Set rtdx = CreateObject("RTDX")
*****
' Open a channel for reading;
' the parameter name in this example must be "ochan."
status = rtdx.Open("ochan", "R")
*****
If status <> Success Then
Debug.Print "Opening of channel ochan failed"
Goto Error_Handler
End If

Do
*****
' Read a 32-bit integer from the target.
*****
status = rtdx.ReadId(data)

Select Case (status)
Case Success
If status = Success Then
Debug.Print "Value " & _
data & _
" was received from the target"
End If
Case ENODataAvailable
Debug.Print "No data is currently available"
Case ENODflogFile
Debug.Print "End of log file has been detected"
Case Failure
Debug.Print "ReadId returned failure"
Exit Do
Case Else
Debug.Print "Unknown return code"
Exit Do
End Select
Loop Until status = ENODflogFile
' Close the channel.
*****
status = rtdx.Close()
*****
' Release the reference to the RTDX COM object.
*****
Set rtdx = Nothing

Exit Sub

Error_Handler:

```

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007

```
Debug.Print "Error in COM method call"
Set rtdx = Nothing

End Sub
```

### Visual C++ Host Client Code S2L2.cpp

RTDX-S2

The following is the finished Visual C++ version of the Visual Basic code S2L2.bas.

#### Complete Code

```
//#####
// S2L2.cpp - (Section 2, Lesson 2) RECEIVING AN INTEGER FROM THE
// TARGET
// This is the RTDX COM Client for Section 2, Lesson 2
//
// This example receives integer value 5 from the target
//#####
#include <iostream.h>
using namespace RTDXINTLib;

int main()
{
    IRtdkExpPtr rtdx; // holds pointer to IRtdkExp interface
    long data; // holds data received from target application

```

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhB A53.htm

26/09/2007

Getting Started With the Code Composer Studio Tutorial

Page 490 sur 548

```
long status; // holds status of RTDX COM API calls
HRESULT hr; // holds status of generic COM API calls

// initialize COM
::CoInitialize( NULL );

// instantiate the RTDX COM Object
hr = rtdx.CreateInstance( L"RTDX" );

cout.setf( ios::showbase );

if ( FAILED(hr) ) {
    cerr << hex << hr << " - Error: instantiation failed! \n";
    return -1;
}

// open a channel (ochan) for reading data
status = rtdx->Open( "ochan", "R" );

if ( status != Succeeded ) {
    cerr << hex << status \

<< " - Error: Opening of channel \"ochan\" failed! \n";
    return -1;
}

// loop until we have read all of our messages
do {
    // read a message

```

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhB A53.htm

26/09/2007

```
status = rtdx->ReadId( kdata );  
// test status returned from ReadId  
switch ( status ) {  
    case Success:  
        // display data  
        cout << data << "\n";  
        break;  
    case Failure:  
        cerr << hex << status \  
        << " - Error: ReadId returned failure! \n";  
        return -1;  
    case ENoDataAvailable:  
        cout << "\nNo Data is currently available!\n";  
        cout << "\nWould you like to continue reading [y or n]?\n";  
        char option;  
        cin >> option;  
        if ( option == 'Y' ) || (option == 'y' )  
            break;  
        else  
            &#9;:&#9;:&#9;:&#9;return -1;  
        case EEndOfFile:  

```

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhB A53.htm

26/09/2007

Getting Started With the Code Composer Studio Tutorial

Page 492 sur 548

```
cout << "\nData Processing Complete!\n";  
break;  
default:  
    cerr << hex << status \  
    << " - Error: Unknown return code! \n";  
    return -1;  
}  
} while ( status != EEndOfFile );  
// close the channel  
status = rtdx->Close();  
// release the RTDX COM Object  
rtdx.Release();  
// unitialize COM  
::CoUninitialize();  
return 0;  
}
```

**Visual Basic Code SZL3.bas**

RTDX-S2

The following is the finished Visual Basic code SZL3.bas.

Complete Code

Option Explicit

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhB A53.htm

26/09/2007

```

*****
** SZ13 has - (Section 2, Lesson 3) RECEIVING ENTIRE MESSAGES USING
** SAFERARRAYS
** This is the RTDX Host Client for Section 2, Lesson 3
**
** This example receives integer values 1-10 from the target
*****
Const Success = #H0 ' Method call is valid
Const Failure = #H80004005 ' Method call failed

Const ENODATAAVAILABLE = #H8003001E ' No data was available.
' However, more data may be
' available in the future.

Const ERMDFLOGFILE = #H80030002 ' No data was available.
' The end of the log file has
' been reached.

Sub main()

Dim rtdx As Object
*****
' Declare a variable of type Variant. This variable contains
' a pointer to a SAFERARRAY.
*****
Dim vardata As Variant

Dim status As Long
Dim i As Long

On Error GoTo Error_Handler

' Create an instance of the RTDX COM object
Set rtdx = CreateObject("RTDX")

```

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhB A53.htm

26/09/2007

Getting Started With the Code Composer Studio Tutorial

Page 494 sur 548

```

' Open channel ochan for reading
status = rtdx.Open("ochan", "R")

If status <> Success Then
Debug.Print "Opening of channel ochan failed"
Goto Error_Handler
End If

Do
*****
' Read a message (32-bit integer array) from the target.
*****
status = rtdx.ReadSZ14(vardata)

Select Case (status)
Case Success
*****
*****
' Print each element in the array.
*****
For i = Lbound(vardata) To Ubound(vardata)
Debug.Print vardata(i)
Next i
Case ENODATAAVAILABLE
Debug.Print "No data is currently available"
Case ERMDFLOGFILE
Debug.Print "End of log file has been detected"
Case Failure
Debug.Print "ReadSZ14 returned failure"
Exit Do
Case Else
Debug.Print "Unknown return code"
Exit Do
End Select
Loop Until status = ERMDFLOGFILE

' Close the channel
status = rtdx.Close()

' Release the reference to the RTDX COM object
Set rtdx = Nothing

Exit Sub

Error_Handler:

Debug.Print "Error in COM method call"

```

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhB A53.htm

26/09/2007

```
Set rtdx = Nothing
End Sub
```

### Visual C++ Host Client Code S2L3.cpp

RTDX-S2

The following is the finished Visual C++ version of the Visual Basic code S2L3.bas.

#### Complete Code

```
//#####
// S2L3.cpp - (Section 2, Lesson 3) RECEIVING ENTIRE MESSAGES USING
// SAFERARRAYS
// This is the RTDX Host Client for Section 2, Lesson 3
//
// This example receives integer values 1-10 from the target
//#####
#import "c:\ccstudio\cc\bin\rtdxint.dll"
#include <iostream.h>
using namespace RTDXINTLib;
int main()
{
    IRcdkExpPtr rtdx; // holds pointer to IRcdkExp interface
    long status; // holds status of RTDX COM API calls
```

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhB A53.htm

26/09/2007

Getting Started With the Code Composer Studio Tutorial

Page 496 sur 548

```
HRESULT hr; // holds status of generic COM API calls
VARIANT sai; // holds pointer to SAFERARRAY
long data; // holds data from SAFERARRAY
long i; // SAFERARRAY incrementer
// initialize COM
::CoInitialize( NULL );
// initialize VARIANT
::VariantInit( &sai );
// instantiate the RTDX COM Object
hr = rtdx.CreateInstance( __uuidof(RTDXINTLib::RcdkExp) );
cout.setf( ios::showbase );
if ( FAILED(hr) ) {
    cerr << hex << hr << " - Error: Instantiation failed! \n";
    return -1;
}
// open a channel (ochan) for reading
status = rtdx->Open( "ochan", "R" );
if ( status != Succeeded ) {
    cerr << hex << status \
    << " - Error: Opening of a channel failed! \n";
    return -1;
}
```

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhB A53.htm

26/09/2007



```
    )
    // Loop until we have read all of our messages
    do {
        // read a 32-bit integer message
        status = rtdx->ReadSMT4( ksa );
        // test status returned from ReadSMT4
        switch ( status ) {
            case Success:
                // Display data
                cout << "\n";
                for ( i = 0; i < (signed)sa.parray->rgsaabound[0].cElements; i++) {
                    hr = ::SafeArrayGetElement( sa.parray, &i, (long*)&data );
                    cout << data << "\t";
                }
                break;
            case Failure:
                cerr << hex \
                    << status \
                    << " - Error: ReadSMT4 returned failure! \n";
                return -1;
            case ENoDataAvailable:
                return -1;
        }
    }
```

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhB A53.htm

26/09/2007

Getting Started With the Code Composer Studio Tutorial

Page 498 sur 548

```
    cout << "\n\nNo Data is currently available!\n";
    cout << "\n\nWould you like to continue reading [Y or n]?\n";
    char option;
    cin >> option;
    if ( (option == 'Y') || (option == 'y') )
        break;
    else
        return -1;
    case ENoLogFile:
        cout << "\n\nData Processing Complete!\n";
        break;
    default:
        cerr << hex << status << " - Error: Unknown return code! \n";
        return -1;
    }
    } while ( status != ENoLogFile );
    // close the channel
    status = rtdx->Close();
    // release the RTDX COM Object
    rtdx.Release();
    // Clear Variant
```

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhB A53.htm

26/09/2007

```

::VariantClear( &sa );
// initialize COM
::CoInitialize();
return 0;
}

```

### Visual Basic Code SZL4.bas

RTDX-S2

The following is the finished Visual Basic code SZL4.bas.

Complete Code

Option Explicit

```

*****
** SZL4.bas - (Section 2, Lesson 4) SENDING AN INTEGER TO THE TARGET
** This is the RTDX Host Client for Section 2, Lesson 4
**
** This example sends integer value 5 to the target
*****
Const Success = &H0 ' Method call is valid
Const Failure = &H80004005 ' Method call failed

Const ENODATAAVAILABLE = &H8003001E ' No data was available.
' However, more data may be
' available in the future.

```

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007

Getting Started With the Code Composer Studio Tutorial

Page 500 sur 548

```

Const ERRNOFLAGFILE = &H80030002 ' No data was available.
' The end of the log file has
' been reached.

Const VALUE_TO_SEND = 5

Sub main()

Dim rtdx As Object
Dim data As Long
Dim bufferstate As Long
Dim status As Long

On Error GoTo Error_Handler

' Create an instance of the RTDX COM object
Set rtdx = CreateObject("RTDX")

' Open channel ichan for writing
status = rtdx.Open("ichan", "W")

If status <> Success Then
Debug.Print "Opening of channel ichan failed"
GoTo Error_Handler
End If

data = VALUE_TO_SEND
*****
' Send a 32-bit integer to the target.
*****
status = rtdx.Write14(data, bufferstate)

If status = Success Then
Debug.Print "Value " & data & " was sent to the target"
Else
Debug.Print "Write14 failed"
End If

' Close the channel
status = rtdx.Close()

' Release the reference to the RTDX COM object
Set rtdx = Nothing

Exit Sub

```

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007

```

Error_Handler:
Debug.Print("Error in COM method call")

Set rtdx = Nothing

End Sub

```

### Visual C++ Host Client Code SZL4.cpp

RTDX-S2

The following is the finished Visual C++ version of the Visual Basic code SZL4.bas.

```

Complete Code
#####
// SZL4.cpp - (Section 2, Lesson 4) SENDING AN INTEGER TO THE TARGET
// This is the RTDX Host Client for Section 2, Lesson 4
// This example sends integer value 5 to the target
#####
#include "c:\ccstudio\ccbin\rdxint.dll"
#define VALUE_TO_SEND 5
using namespace RTDXINTLib;

int main()
{
    IRtdxExpPr rdx; // holds pointer to IRtdxExp interface
    long status; // holds status of RTDX COM API calls
    HRESULT hr; // holds status of generic COM API calls
    long data=VALUE_TO_SEND; // holds data to be sent to target
    long bufferSize; // holds the state of the host's write buffer

    // initialize COM
    ::CoInitialize( NULL );

    // instantiate the RTDX COM Object
    hr = rdx.CreateInstance( L"RTDX" );

    cout.setf( ios::showbase );

    if ( FAILED(hr) ) {

```

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007

Getting Started With the Code Composer Studio Tutorial

Page 502 sur 548

```

    cerr << hex << hr << " - Error: Instantiation failed!\n";
    return -1;
}

// open a channel (chan) for writing
status = rdx->Open( "chan", "w" );

if ( status != Success ) {
    cerr << hex << status \
    << " - Error: Opening of channel \"chan\" failed!\n";
    return -1;
}

// send a 32-bit integer to the target
rdx->Write4( data, &bufferstate );

if ( status != Success ) {
    cerr << hex << status << " - Error: Write4 failed!\n";
    return -1;
}

cout << "Value " << data << " was sent to the target!\n";

// close the channel
status = rdx->Close();

// release the RTDX COM Object
rdx.Release();

// uninitialize COM
::CoUninitialize();

return 0;
}

```

### Visual Basic Code SZL5.bas

RTDX-S2

The following is the finished Visual Basic code SZL5.bas.

```

Complete Code

Option Explicit

*****

```

```

** SZL5.bas - (Section 2, Lesson 5) SENDING AN ARRAY OF INTEGERS USING

```

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007

```

** SAFERARRAYS
** This is the RTDX Host Client for Section 2, Lesson 5
**
** This example sends integer values 1-10 to the target
*****
Const Success = &H0 ' Method call is valid
Const Failure = &H80004005 ' Method call failed
Const ENODATAAvailable = &H8003001E ' No data was available.
' However, more data may be
' available in the future.
Const ENDOfFile = &H80030002 ' No data was available.
' The end of the log file has
' been reached.

Sub main()
Dim rtdx As Object
Dim arraydata(10) As Long
Dim data As Long
Dim bufferSize As Long
Dim status As Long
Dim i As Long

On Error Goto ErrorHandler

' Create an instance of the RTDX COM object
Set rtdx = CreateObject("RTDX")

' Open channel iChan for writing
status = rtdx.Open("iChan", "W")

If status <> Success Then
Debug.Print "Opening of channel iChan failed"
Goto ErrorHandler
End If

' Fill up a SAFERARRAY with values to send to the
' target.
*****
data = 1
For i = LBound(arraydata) To (UBound(arraydata) - 1)
arraydata(i) = data
data = data + 1
Next i
*****
' Write a SAFERARRAY of 32-bit integers to the target.
*****
status = rtdx.Write(CVar(arraydata), bufferSize)
If status = Success Then
For i = LBound(arraydata) To (UBound(arraydata) - 1)
Debug.Print "Value " & arraydata(i) & " was sent to the target."
Next i
Else
Debug.Print "Writei4 failed"
End If
' Close the channel
status = rtdx.Close()
' Release the RTDX COM object
Set rtdx = Nothing
Exit Sub
ErrorHandler:
Debug.Print "Error in COM method call"
Set rtdx = Nothing
End Sub

```

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhB A53.htm

26/09/2007

Getting Started With the Code Composer Studio Tutorial

Page 504 sur 548

```

*****
' Fill up a SAFERARRAY with values to send to the
' target.
*****
data = 1
For i = LBound(arraydata) To (UBound(arraydata) - 1)
arraydata(i) = data
data = data + 1
Next i
*****
' Write a SAFERARRAY of 32-bit integers to the target.
*****
status = rtdx.Write(CVar(arraydata), bufferSize)
If status = Success Then
For i = LBound(arraydata) To (UBound(arraydata) - 1)
Debug.Print "Value " & arraydata(i) & " was sent to the target."
Next i
Else
Debug.Print "Writei4 failed"
End If
' Close the channel
status = rtdx.Close()
' Release the RTDX COM object
Set rtdx = Nothing
Exit Sub
ErrorHandler:
Debug.Print "Error in COM method call"
Set rtdx = Nothing
End Sub

```

---

**Visual C++ Host Client Code S2L5.cpp**

RTDX-S2

The following is the finished Visual C++ version of the Visual Basic code S2L5.bas.

Complete Code

//#####

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhB A53.htm

26/09/2007

```
// S2L5.cpp - (Section 2, Lesson 5) SENDING AN ARRAY OF INTEGERS
// USING SAFEARRAYS
// This is the RTDX Host Client for Section 2, Lesson 5
//
// This example sends integer values 1-10 to the target
#####
#import "c:\ccstudio\cc\bin\rtdxint.dll"
#include <iostream.h>
#define MAX_ELEMENTS 10;
using namespace RTDXINTLib;

int main()
{
    IRtdxExpPtr rdx; // holds pointer to IRtdxExp interface
    long status; // holds status of RTDX COM API calls
    HRESULT hr; // holds status of generic COM API calls
    VARIANT sa; // holds a pointer to a SAFEARRAY
    SAFEARRAYBOUND rgsabound[1]; // set the dimension of the SAFEARRAY
    long bufferstate; // holds the state of the host's write buffer
    long data; // holds the element data of SAFEARRAY
```

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007

Getting Started With the Code Composer Studio Tutorial

Page 506 sur 548

```
// Initialize COM
::CoInitialize( NULL );

// Initialize VARIANT
::VariantInit( &sa );

// instantiate the RTDX COM Object
hr = rdx.CreateInstance( __uuidof(RTDXINTLib::RtdxExp) );

cout.setf(ios::showbase);

if ( FAILED(hr) ) {
    cerr << hex << hr << " - Error: Instantiation failed! \n";
    return -1;
}

// open a channel (chan) for writing
status = rdx->Open( "chan", "w" );

if ( status != Success ) {
    cerr << hex << status \
```

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007

```
<< " - Error: Opening of channel \"chan\" failed! \n";
return -1;
}

// set the VARIANT to be a SAFEARRAY of 32-bit integers
sa.vt = VT_ARRAY | VT_I4;

// set the lower bound of the SAFEARRAY
rgsabound[0].lbound = 0;

// set the number of elements in the SAFEARRAY
rgsabound[0].cElements = MAX_ELEMENTS;

// create the SAFEARRAY
sa.parray = SafeArrayCreate( VT_I4, 1, rgsabound );

// fill up the SAFEARRAY with values
data = 1;
for (long i = 0; i < (signed)sa.parray->rgsabound[0].cElements; i++) {
    hr = ::SafeArrayPutElement( sa.parray, &i, (long*)&data);
    data++;
}
```

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007

Getting Started With the Code Composer Studio Tutorial

Page 508 sur 548

```
}

// send data to the target
status = rtdx->Write(sa, &bufferstate);

if (status != Success) {
    cerr << hex << status << " - Error: Write failed!\n";
    return -1;
}

// print out data
for (long j=0; j<(signed)sa.parray->rgsabound[0].cElements; j++) {
    hr = ::SafeArrayGetElement( sa.parray, &j, (long*)&data);
    cout << "Value " << data << " was sent to the target!\n";
}

// close the channel
status = rtdx->Close();

// release the RTDX COM Object
rtdx.Release();
```

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007

```
// clear VARIANT
::VariantClear(&sa);

// unitalize COM
::CoUninitialize();

return 0;
}
}
```

### Visual Basic Code SZL6.bas

RTDX-S2

The following is the finished Visual Basic code SZL6.bas.

```
Complete Code
Option Explicit

*****
** SZL6.bas - (Section 2, Lesson 6) RECEIVING DATA FROM MULTIPLE
** CHANNELS
** This is the RTDX Host Client for Section 2, Lesson 6
**
** This example receives integer values:
** 3 from channel ochan1
```

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhB A53.htm

26/09/2007

Getting Started With the Code Composer Studio Tutorial

Page 510 sur 548

```
** 2 from channel ochan2
** 1 from channel ochan3
*****
Const Success = &H0 ' Method call is valid
Const Failure = &H80004005 ' Method call failed
Const ENGDDataAvailable = &H8003001E ' No data was available.
' However, more data may be
' available in the future.
Const EndOfFile = &H80030002 ' No data was available.
' The end of the log file has
' been reached.
*****
' Create a structure to hold the channel attributes.
*****
Public Type RTDX_Channel
    obj As Object
    name As String
    type As String
    opened As Boolean
End Type

Sub main()
*****
' Create an array instance of the type RTDX_Channel.
*****
Dim rtdx(1 To 3) As RTDX_Channel
```

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhB A53.htm

26/09/2007

```

Dim data As Long
Dim status As Long
Dim i As Long

On Error GoTo Error_Handler
'*****
' Initialize the channel name:
' the parameter names in this example must be "ochan1,"
' "ochan2," and "ochan3."
'*****
rtdx(1).name = "ochan1"
rtdx(2).name = "ochan2"
rtdx(3).name = "ochan3"
'*****
' Create an instance of the RTDX COM object and open the
' channels.
'*****
For i = LBound(rtdx) To UBound(rtdx)
    rtdx(i).type = "R"
    rtdx(i).opened = False
    Set rtdx(i).obj = CreateObject("RTDX")
    status = rtdx(i).obj.Open(rtdx(i).name, _
        rtdx(i).type)
    If status <> Success Then
        Debug.Print "Opening of channel " & _
            rtdx(i).name & _
            " failed" & _
            Chr(10)
    Else
        rtdx(i).opened = True
    End If
Next i

For i = LBound(rtdx) To UBound(rtdx)
    If rtdx(i).opened Then
        '*****
        ' Read a 32-bit integer from the target.
        '*****
        status = rtdx(i).obj.Read4(data)

        Select Case (status)

            Case Success
                Debug.Print "Value " & _
                    data & _
                    " was received from target via channel " & _
                    rtdx(i).name
            Case Error
                Debug.Print "No data is currently available"
            Case ErrObject
                Debug.Print "End of log file has been detected"
            rtdx(i).opened = False
            Case Failure
        End Select
    End If
Next i

```

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007

Getting Started With the Code Composer Studio Tutorial

Page 512 sur 548

```

Debug.Print "Read4 returned failure"
rtdx(i).opened = False
Case Else
    Debug.Print "Unknown return code"
Exit For
End Select
Next i

'*****
' Close the channel and release the reference to the RTDX
' COM object.
'*****
For i = LBound(rtdx) To UBound(rtdx)
    status = rtdx(i).obj.Close()
    Set rtdx(i).obj = Nothing
Next i
Exit Sub

Error_Handler:
Debug.Print "Error in COM method call"

' Release the reference to the RTDX COM object
For i = LBound(rtdx) To UBound(rtdx)
    Set rtdx(i).obj = Nothing
Next i
End Sub

```

### Visual Basic Code SZL7.bas

RTDX-S2

The following is the finished Visual Basic code SZL7.bas.

Complete Code  
Option Explicit

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007



```

*****
** S2L7_bas - (Section 2, Lesson 7) RECEIVING DATA FROM THE ALL CHANNEL
**
** This is the RTDX Host Client for Section 2, Lesson 7
**
** This example receives integer values 1-3 from the target via the
** ALL channel
*****
Const Success = #H0 ' Method call is valid
Const Failure = #H80004005 ' Method call failed

Const EMODataAvailable = #H8003001E ' No data was available.
' However, more data may be
' available in the future.

Const EMODataFile = #H80030002 ' No data was available.
' The end of the log file has
' been reached.

Sub main()

Dim rtdx As Object
Dim data As Long
Dim status As Long
Dim i As Long

On Error Goto Error_Handler

' Create an instance of the RTDX COM object
Set rtdx = CreateObject("RTDX")

*****
' Open the "ALL" channel for reading.
*****
status = rtdx.Open("ALL", "R")

```

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007

Getting Started With the Code Composer Studio Tutorial

Page 514 sur 548

```

If status <> Success Then
Debug.Print "Opening of channel ALL failed"
Goto Error_Handler
End If

Do
' Read a 32-bit integer from the target
status = rtdx.Read1(data)

Select Case (status)
Case Success
If status = Success Then
Debug.Print "Value " & _
data & _
" was received from the target"
End If
Case EMODataAvailable
Debug.Print "No data is currently available"
Case EMODataFile
Debug.Print "End of log file has been detected"
Case Failure
Debug.Print "Read1 returned failure"
Exit Do
Case Else
Debug.Print "Unknown return code"
Exit Do
End Select
Loop Until status = EMODataFile

' Close the channel
status = rtdx.Close()

' Release the reference to the RTDX COM object
Set rtdx = Nothing

Exit Sub

Error_Handler:

Debug.Print "Error in COM method call"

Set rtdx = Nothing

End Sub

```

Visual Basic Code S2L8.bas

RTDX-S2

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007

```

The following is the finished Visual Basic code SZL8.bas.

Complete Code
Option Explicit

'*****
'* SZL8.bas - (Section 2, Lesson 8) SEEKING THROUGH MESSAGES
'* This is the RTDX Host Client for Section 2, Lesson 8
'*
'* This example receives integer values 1-10 from the target, then uses
'* the Seek() method to read the values in reverse order.
'*****
Const Success = &H0 ' Method call is valid
Const Failure = &H80004005 ' Method call failed
Const ENoDataAvailable = &H8003001E ' No data was available.
' However, more data may be
' available in the future.
Const EEndOfLogFile = &H80030002 ' No data was available.
' The end of the log file has
' been reached.

Sub main()

Dim rdx As Object
Dim data As Long
Dim status As Long
Dim nummsg As Long
Dim i As Long

' Create an instance of the RTDX COM object
Set rdx = CreateObject("RTDX")

' Open channel ochan for reading
status = rdx.Open("ochan", "R")

If status <> Success Then
Debug.Print "Opening of channel ochan failed"
Goto Error_Handler
End If

'*****
' Get the total number of messages received by the host.
'*****
status = rdx.GetNumMsgs(nummsg)

If status <> Success Then
Debug.Print "GetNumMsgs returned failure"
Goto Error_Handler
End If

' Read data from beginning to end
Debug.Print "Reading data from beginning to end"
For i = 1 To nummsg
' Read a 32-bit integer from the target
status = rdx.Read4(data)
Select Case (status)
Case Success
Debug.Print "Value " & _
data & _
" was received from the target"
Case ENoDataAvailable
Debug.Print "No data is currently available"
Case Failure
Debug.Print "Read4 returned failure"
Exit For
Case Else
Debug.Print "Unknown return code"
Exit For
End Select
Next i

' Rewind the log of data.
'*****
status = rdx.Rewind()

If status <> Success Then
Debug.Print "Rewind returned failure"
Goto Error_Handler

```

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007

Getting Started With the Code Composer Studio Tutorial

Page 516 sur 548

```

On Error Goto Error_Handler

' Create an instance of the RTDX COM object
Set rdx = CreateObject("RTDX")

' Open channel ochan for reading
status = rdx.Open("ochan", "R")

If status <> Success Then
Debug.Print "Opening of channel ochan failed"
Goto Error_Handler
End If

'*****
' Get the total number of messages received by the host.
'*****
status = rdx.GetNumMsgs(nummsg)

If status <> Success Then
Debug.Print "GetNumMsgs returned failure"
Goto Error_Handler
End If

' Read data from beginning to end
Debug.Print "Reading data from beginning to end"
For i = 1 To nummsg
' Read a 32-bit integer from the target
status = rdx.Read4(data)
Select Case (status)
Case Success
Debug.Print "Value " & _
data & _
" was received from the target"
Case ENoDataAvailable
Debug.Print "No data is currently available"
Case Failure
Debug.Print "Read4 returned failure"
Exit For
Case Else
Debug.Print "Unknown return code"
Exit For
End Select
Next i

' Rewind the log of data.
'*****
status = rdx.Rewind()

If status <> Success Then
Debug.Print "Rewind returned failure"
Goto Error_Handler

```

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007

```
End If
' Read data from end to beginning
Debug Print Chr(10) & "Reading data from end to beginning"
For i = nummsgs To 1 Step -1
*****
' Seek to a specific message.
*****
status = rtdx.Seek(i)

If status <> Success Then
Debug Print "Seek returned failure"
Goto Error_Handler
End If

' Read a 32-bit integer from the target
status = rtdx.Read4(data)
Select Case (status)
Case Is = Success
Debug Print "Value " & _
data & _
" was received from the target" & _
Chr(10)
Case Is = ENoDataAvailable
Debug Print "No data is currently available"
Case Is = Failure
Debug Print "Read4 returned failure"
Exit For
Case Else
Debug Print "Unknown return code"
Exit For
End Select
Next i

' Close the channel
status = rtdx.Close()

' Release the reference to the RTDX COM object
Set rtdx = Nothing
Exit Sub

Error_Handler:
Debug Print "Error in COM method call"

Set rtdx = Nothing
```

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhB A53.htm

26/09/2007

Getting Started With the Code Composer Studio Tutorial

Page 518 sur 548

```
End Sub

' Visual Basic Code S2L9.bas

The following is the finished Visual Basic code S2L9.bas.

Complete Code

Option Explicit

*****
'* S2L9.bas - (Section 2, Lesson 9) REMOTE ENABLING AND DISABLING OF A
'* CHANNEL
'*
'* This is the RTDX Host Client for Section 2, Lesson 9
'*
'* This example enables the target channel (chan), receives integer
'* value 5 from the target, and disables the channel
*****
Const Success = &H0 ' Method call is valid
Const Failure = &H80004005 ' Method call failed
Const ENoDataAvailable = &H8003001E ' No data was available.
' However, more data may be
' available in the future.
Const EEndOfFile = &H80030002 ' No data was available.

' The end of the log file has
' been reached.
```

RTDX-S2

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhB A53.htm

26/09/2007

```

Const CHANNEL_NAME = "ochan"

Sub main()

Dim rtdx As Object
Dim data As Long
Dim status As Long
Dim nummsg As Long
Dim i As Long

On Error Goto Error_Handler

Set rtdx = CreateObject("RTDX")

'*****
' Enable channel for target writes
'*****
status = rtdx.EnableChannel(CHANNEL_NAME)

If status <> Success Then
Debug.Print "enabling of channel ochan failed"
Goto Error_Handler
End If

status = rtdx.Open(CHANNEL_NAME, "R")

If status <> Success Then
Debug.Print "Opening of channel ochan failed"
Goto Error_Handler
End If

status = rtdx.GetNumMsgs(nummsg)

If status <> Success Then
Debug.Print "GetNumMsgs returned failure"
Goto Error_Handler
End If

For i = 1 To nummsg
status = rtdx.ReadId(data)
Select Case (status)
Case Success
Debug.Print "Value " & _
data & _

```

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhB A53.htm

26/09/2007

Getting Started With the Code Composer Studio Tutorial

Page 520 sur 548

```

" was received from the target " & _
Chr(10)
Case ENoDataAvailable
Debug.Print "No data is currently available"
Case EEndOfFile
Debug.Print "End of log file has been detected"
Case Failure
Debug.Print "ReadId returned failure"
Exit For
Case Else
Debug.Print "Unknown return code"
Exit For
End Select
Next i

'*****
' Disable channel from target writes.
'*****
status = rtdx.DisableChannel(CHANNEL_NAME)

status = rtdx.Close()

If status <> Success Then
Debug.Print "enabling of channel ochan failed"
Goto Error_Handler
End If

Set rtdx = Nothing

Exit Sub

Error_Handler:
Debug.Print "Error in COM method call"

Set rtdx = Nothing

End Sub

```

### Visual Basic Code S4L2.bas

RTDX-S4

The following is the finished Visual Basic code S4L2.bas.

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhB A53.htm

26/09/2007

```

Complete Code
Option Explicit
'*****
'
' * S4L2.bas - (Section 4, Lesson 2) USING THE FLUSH ROUTINE
'
' * This is the RTDX Host Client for Section 4, Lesson 2
'
' *
' * This example sends integer value 5 to the target and then flushes
' * the channel.
'*****
Const Success = &H0 ' Method call is valid
Const Failure = &H80004005 ' Method call failed

Const ENoDataAvailable = &H8003001E ' No data was available.
' However, more data may be
' available in the future.
Const EEndOfFile = &H80030002 ' No data was available.
' The end of the log file has
' been reached.

Const VALUE_TO_SEND = 5

Sub main()

Dim rdx As Object
Dim data As Long
Dim bufferstate As Long
Dim status As Long

'*****
'
' * This example sends integer value 5 to the target and then flushes
' * the channel.
'*****
Const Success = &H0 ' Method call is valid
Const Failure = &H80004005 ' Method call failed

Const ENoDataAvailable = &H8003001E ' No data was available.
' However, more data may be
' available in the future.
Const EEndOfFile = &H80030002 ' No data was available.
' The end of the log file has
' been reached.

Const VALUE_TO_SEND = 5

Sub main()

Dim rdx As Object
Dim data As Long
Dim bufferstate As Long
Dim status As Long

```

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007

Getting Started With the Code Composer Studio Tutorial

Page 522 sur 548

```

On Error GoTo Error_Handler

' Create an instance of the RTDX COM object
Set rdx = CreateObject("RTDX")

' Open channel (chan for writing
status = rdx.Open("chan", "W")

If status <> Success Then
Debug.Print "Opening of channel (chan failed"
Goto Error_Handler
End If

data = VALUE_TO_SEND
' send a 32-bit integer to the target
status = rdx.Write14(data, bufferstate)

'*****
'
' * Verify if target application has requested data from the host.
'*****
status = rdx.StatusOnWrite(bufferstate)

If status <> Success Then
Debug.Print "Error in call to StatusOnWrite"
Goto Error_Handler
End If

If bufferstate <= 0 Then

'*****
'
' * Flush the channel.
'*****
status = rdx.Flush()

If status <> Success Then
Debug.Print "Error in call to Flush"
Goto Error_Handler
End If

If status = Success Then
Debug.Print "Value " & data & " was sent to the target"
Else
Debug.Print "Write14 failed"
End If

' Close the channel
status = rdx.Close()

' Release the reference to the RTDX COM object

```

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007

```

Set rdx = Nothing
Exit Sub

Error_Handler:
Debug.Print "Error in COM method call"
Set rdx = Nothing

End Sub

```

### Target Application C Code S4L3.c

RTDX-S4

The following is the finished code S4L3.c.

Complete Code

```

/*****
 * S4L3.c - (Section 4, Lesson 3) SENDING STRUCTURED DATA
 * This is the RTDX Target Code for Section 4, Lesson 3
 * This example sends a structure of 3 integers, with values 1, 2 & 3
 * to the host.
 *****/

#include <rtdx.h> /* defines RTDX target API calls */
#include "target.h" /* defines TARGET_INITIALIZE() */
#include <stdio.h> /* C_I/O */

/* These are the values we are going to send to the host */
#define VALUE_FOR_SDATA1 1

```

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007

Getting Started With the Code Composer Studio Tutorial

Page 524 sur 548

```

/*****
 * Declare a structure to be sent to the host;
 * In this example, you must declare a structure of 3 integers,
 * with values 1, 2, and 3.
 *****/

struct {
    int s_data1;
    int s_data2;
    int s_data3;
}mystruct;

/* declare a global output an channel */
RTDX_CreateOutputChannel( ochan );

void main()
{
    int status;

    mystruct.s_data1 = VALUE_FOR_SDATA1;
    mystruct.s_data2 = VALUE_FOR_SDATA2;
    mystruct.s_data3 = VALUE_FOR_SDATA3;
    TARGET_INITIALIZE();

    /* enable the output channel */
    RTDX_enableOutput( ochan );

```

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007

```
/* Send the structure to the host.
*****
status = RTDX_write( &ochan, &mystruct, sizeof(mystruct) );
if ( status == 0 ) {
    puts( "ERROR: RTDX_write failed!\n" );
    exit( -1 );
}
while ( RTDX_write != NULL ) {
    #if RTDX_POLLING_IMPLEMENTATION
    RTDX_poll();
    #endif
}
}
/* disable the output channel */
RTDX_disableOutput( &ochan );

puts( "Program Complete!\n" );
}
```

### Visual Basic Code S4L6.bas

RTDX-S4

The following is the finished Visual Basic code S4L6.bas.

Complete Code

Option Explicit

\*\*\*\*\*

/\* S4L6.bas - (Section 4, Lesson 6) USING THE SETPROCESSOR ROUTINE

/\* This is the RTDX Host Client for Section 4, Lesson 6

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhB A53.htm

26/09/2007

Getting Started With the Code Composer Studio Tutorial

Page 526 sur 548

```
/*
*****
/* This example receives integer values 1-10 from the target
*****
Const Success = &H0 ' Method call is valid
Const Failure = &H80004005 ' Method call failed
Const ENODATAAvailable = &H8003001E ' No data was available.
' However, more data may be
' available in the future.
Const ECHNOLogFile = &H80030002 ' No data was available
' The end of the log file has
' been reached.
```

Sub main()

Dim rtdx As Object

Dim vardata As Variant

Dim status As Long

Dim i As Long

Dim board As String

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhB A53.htm

26/09/2007

```
Dim cpu As String

On Error GoTo Error_Handler

' Create an instance of the RTDX COM object
Set rdx = CreateObject("RTDX")

*****

' Specify the board and processor.
*****

board = "C6xxx EVM (Texas Instruments)"
cpu = "TMS320C6701EVM"

*****

' Attach to the specified board and processor.
*****

status = rdx.SetProcessor(board, cpu)

If status <> Success Then
```

file:///C:/Documents and Settings/Pierre-Armand/Local Settings/Temp/~hhBA53.htm

26/09/2007

Getting Started With the Code Composer Studio Tutorial

Page 528 sur 548

```
Debug.Print "Attempt to attach to specified processor failed"

Goto Error_Handler

End If

' Open channel ochan for reading
status = rdx.Open("ochan", "R")

If status <> Success Then

Debug.Print "Opening of channel ochan failed"

Goto Error_Handler

End If

Do

' read a message (32-bit integer array) from the target
status = rdx.ReadSA14(vardata)

Select Case (status)
Case Success

' print each element in the array
For i = Lbound(vardata) To Ubound(vardata)
```

file:///C:/Documents and Settings/Pierre-Armand/Local Settings/Temp/~hhBA53.htm

26/09/2007



```
Debug.Print vardata()
Next !
Case ENODATAAVAILABLE
Debug.Print "No data is currently available"
Case EENDLOGFILE
Debug.Print "End of log file has been detected"
Case FAILURE
Debug.Print "ReadSAL4 returned failure"
Exit Do
Case Else
Debug.Print "Unknown return code"
Exit Do
End Select
Loop Until status = EENDLOGFILE

' Close the channel
status = rtdx.Close()

' Release the reference to the RTDX COM object
Set rtdx = Nothing
```

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007

Getting Started With the Code Composer Studio Tutorial

Page 530 sur 548

```
Exit Sub

Error_Handler:
Debug.Print "Error in COM method call"
Set rtdx = Nothing

End Sub
```

### Visual C++ Host Client Code S4L6.cpp

RTDX-S4

```
The following is the finished Visual C++ version of the Visual Basic code S4L6.vbs.

Complete Code
#####
// S4L6.cpp - (Section 4, Lesson 6) USING THE SETPROCESSOR ROUTINE
// This is the RTDX Host Client for Section 4, Lesson 6
//
// This example receives integer values 1-10 from the target
#####
#import "c:\ccstudio\c\bin\rtdxint.dll"
#include <iostream.h>
```

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007

```
using namespace RTDXINTLib;

int main()
{
    IRtdxExpPr rtdx; // holds pointer to IRtdxExp interface
    long status; // holds status of RTDX COM API calls
    HRESULT hr; // holds status of generic COM API calls
    VARIANT sa; // holds pointer to SAFARRAY
    long data; // holds data from SAFARRAY
    long i; // SAFARRAY incrementer
    BSTR board = ::SysAllocString(L"C6xxx EVM (Texas Instruments)");
    BSTR cpu = ::SysAllocString(L"CPU_1");

    // initialize COM
    ::CoInitialize( NULL );

    // initialize VARIANT
    ::VariantInit( &sa );

    // instantiate the RTDX COM Object
```

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007

Getting Started With the Code Composer Studio Tutorial

Page 532 sur 548

```
hr = rtdx.CreateInstance( L"RTDX" );

if ( FAILED(hr) ) {
    cerr << hex << hr << " - Error: Instantiation failed! \n";
    return -1;
}

cout.setf( ios::showbase );

// attach to a specific board and processor
status = rtdx->SetProcessor(board, cpu);
if ( status != Succeeded ) {
    cerr << hex << status \
    << " - Error: Attempt to attach to specified processor failed! \n";
    return -1;
}

::SysFreeString(board);
::SysFreeString(cpu);
```

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007

```
// open a channel (ochan) for reading
status = rtdx->Open("ochan", "R");

if ( status != Success ) {
    cerr << hex << status \
    << " - Error: Opening of a channel failed! \n";
    return -1;
}

// loop until we have read all of our messages
do {
    // read a 32-bit integer message
    status = rtdx->ReadSA14( &sa );

    // test status returned from ReadSA14
    switch ( status ) {
    case Success:
        // Display data
        cout << "\n";
        for ( i = 0; i < (signed)sa.parray->rgsabound[0].clements; i++) {
            hr = ::SafeArrayGetElement( sa.parray, &i, (long*)&data );
        }
    }
```

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007

Getting Started With the Code Composer Studio Tutorial

Page 534 sur 548

```
cout << data << "\n";
}
break;
case Failure:
    cerr << hex \
    << status \
    << " - Error: ReadSA14 returned failure! \n";
    return -1;
case ENoDataAvailable:
    cout << "\n\nNo Data is currently available!\n";
    cout << "\n\nWould you like to continue reading [Y or n]?\n";
    char option;
    cin >> option;
    if ( (option == 'Y') || (option == 'y') )
        break;
    else
        return -1;
case EEndOfFile:
    cout << "\n\nData Processing Complete!\n";
    break;
default:
```

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007

```
cerr << hex << status << " - Error: Unknown return code! \n";
return -1;
}
} while ( status != EEndOfLogfile );
// close the channel
status = rtdx->Close();
// release the RTDX COM Object
rtdx.Release();
// clear Variant
::VariantClear( &sa );
// uninitialized COM
::CoUninitialize();
return 0;
}
```

## Introduction

---

file:///C:/Documents and Settings/Pierre-Armand/Local Settings/Temp/~hhBA53.htm

26/09/2007

Getting Started With the Code Composer Studio Tutorial

Page 536 sur 548



REAL - Introduction

This tutorial demonstrates how to use Code Composer Studio™ IDE's real-time debug features. Real-time debug refers to the ability to control execution of [background code](#) while the processor continues to service [time-critical interrupts](#).

### Learning Objectives:

- Identify uses of stop and real-time modes
- Configure interrupts and set breakpoints in preparation for real-time mode
- Enter and exit real-time mode
- Debug time-critical and non-time-critical ISRs in real-time mode
- Protect time-critical sections of the application code from emulator debug accesses

**Example:** [realtime.pjt](#), [dbgvm.pjt](#)

### Target Configuration:

The C64x Real-Time Tutorial requires an XDS-510 target system and relies on periodic interrupts generated by the timers found on the C64x devices. The Real-Time feature is not available on C6414, C6413, or C6416 Revision 1.0x devices. In CCS Setup Utility, we recommend choosing the C64xx XDS510 Emulator configuration for best performance. The files used for this tutorial are located in [C:/CCStudio\\_v3.10/tutorial/errud4xxxrealtime](#). If you are a new Code Composer user, it is recommended that you go through the [Code Composer Studio IDE Tutorial](#) first.

### Application Objective:

This tutorial uses three Interrupt Service Routines (ISRs), one driven by Timer 0, one driven by Timer 1, and the last driven by Timer 2. Timers 0 and 1 are configured as time-critical interrupts and Timer 2 is configured as a non-time-critical interrupt.

This tutorial contains the following lessons. To go directly to a lesson, click on the link below:

[Terminology](#)

[Configuring The Project](#)

[Launching Real-Time Mode](#)

[Debugging a Non-Time-Critical ISR](#)

file:///C:/Documents and Settings/Pierre-Armand/Local Settings/Temp/~hhBA53.htm

26/09/2007

[Debugging a Time-Critical ISR](#)

[Debug Mask Bit Usage](#)

[Further Information](#)



## Terminology

---



REAL - L1

The following sections provide information that will help you to better understand the information presented in this tutorial.

### Debug Terminology

**Background code:** The body of code that can be halted during debugging because it is not time-critical.

**Foreground code:** The code for time-critical interrupt service routines. These are executed even when background code is halted.

**Debug-halt state:** The state in which the device does not execute background code.

**Time-critical interrupt:** An interrupt that must be serviced even when background code is halted. For example, a time-critical interrupt might service a motor controller or a high-speed timer.

**Debug event:** An action that can result in special debug behavior such as halting the device or pulsing one of the signals, EMU0 or EMU1. Actions could include the decoding of a software breakpoint instruction, the occurrence of an analysis breakpoint/watchpoint, or a request from a host processor.

**Break event:** When a debug event causes the device to enter the debug-halt state.

### Execution Control Modes

An emulator communicating with an in-circuit emulation technology-based target supports two debug modes:

- Stop Mode

file:///C:/Documents and Settings/Pierre-Armand/Local Settings/Temp/~hhBA53.htm

26/09/2007

Getting Started With the Code Composer Studio Tutorial

Page 538 sur 548

- Real-Time Mode

All interrupts are blocked if the target is configured for stop mode debug and the target is the halted. Real-time mode allows time-critical interrupts in the foreground code to be taken while the target is halted in background code. Both execution modes can suspend program execution at break events, such as occurrences of software breakpoint instructions, or specified program-space or data-space accesses.

Real-time debug (RTD) capabilities are provided by the in-circuit emulation technology debug support hardware embedded in the silicon. RTD, in the context of Code Composer Studio™ IDE and in-circuit emulation technology, means that interrupts marked as time-critical will continue to be serviced when the target is halted by a user halt, a breakpoint, or any other debug event.

For more information on these modes, see the online help topics for Real-Time Debug.

### Non-Interruptible Code Space

In architectures with an **unprotected pipeline**, certain code segments (delay slots of a branch) cannot be interrupted without corrupting the program results. Real-time interrupts are blocked when the CPU is in a debug halt state at a point where the code is architecturally non-interruptible (in branch delay slots). If interrupts are disabled by the global interrupt enables, real-time interrupts are also disabled.

When debugging in real-time mode, and the processor is in debug-halt state with one or both of the above conditions true, the processor is stopped in non-interruptible code space. To circumvent this issue, the debug architecture contains logic to optionally force all imprecise halts in real-time to interruptible instruction boundaries with global interrupts enabled. However a **precise debug event** such as a software or hardware breakpoint set in branch delay slots will still cause the CPU to enter into a debug-halt state inside non-interruptible code space.

For more information on this code space, see the online help topics for Real-Time Debug.



## Configuring the Project and Interrupts

---



REAL - L2

This lesson shows you how interrupts are handled during **stop mode**. You learn how to configure interrupts and set breakpoints to get ready for real-time mode. You must not enter real-time mode until after the configuration of the interrupts. You also learn how to view watch windows and CPU registers.

Over the course of the lessons, you will utilize the following types of files:

- .c This file contains source code that provides the main functionality of this project
- .jdt This file contains all of your project build and collection option sets

file:///C:/Documents and Settings/Pierre-Armand/Local Settings/Temp/~hhBA53.htm

26/09/2007

To prepare the project, follow these steps:

1. If you have not already done so, from the Windows Start menu, choose Programs→Texas Instruments→Code Composer Studio 3.1→Code Composer Studio 3.1. (Or, double-click the Code Composer Studio icon on your desktop.)
2. From the Project menu, choose Open.
3. Browse to [C:\CCStudio\\_v3.10\Tutorial\emu64xx\realtime](C:\CCStudio_v3.10\Tutorial\emu64xx\realtime) folder and select `realtime.pjt`.
4. Click Open to open the project.
5. From the Project Menu, choose Build Options.
6. Click on the Linker tab.
7. In the Autonic mode menu item, choose Run-time Autoinitialization (-c) from the drop down list.
8. Click OK to close the dialog box.
9. From the Project Menu, choose Rebuild All.
10. From the File menu, choose Load Program.
11. Navigate to the [C:\CCStudio\\_v3.10\Tutorial\emu64xx\realtime\Debug](C:\CCStudio_v3.10\Tutorial\emu64xx\realtime\Debug) folder and select `realtime.out`.
12. Click Open to load the program.
13. From the Debug menu, choose Go Main. This opens the real-time tutorial code, `realtime.c`.
14. Select the line `OutputData(Result)` on line 90 and press F9. Your line number may vary. This is the breakpoint that we will use.  
The selected line of code now has a red dot next to it, indicating a breakpoint has been set.
15. From the View menu, choose Registers→Core Registers. In the register window, the global interrupt (GIE) bits are displayed separately and should be set to 0.
16. Find the DJER register in the list and double-click on it to open the DJER register dialog box.
17. Set the register value to `0x0000C001`.
18. Click Done to close the dialog box.

<file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm>

26/09/2007

Getting Started With the Code Composer Studio Tutorial

Page 540 sur 548

We will now run the program to completion and view the results in `OutputData(Result)`:

1. From the Debug menu, choose Run. When the processor stops, `OutputData(Results)` is shown as the current PC.  
This breakpoint follows the call to the function `SelectTimeCriticalInterrupts()` and `SetupInterrupts()`. `SetupInterrupts()` is used to configure the timer interrupts and also globally enables interrupts, and sets both GIE and NMIE to 1. `SelectTimeCriticalInterrupts()` configures Timer 0 and Timer 1 as time-critical interrupts and Timer 2 as a non time-critical interrupt.
2. From the View menu, choose Watch Window.
3. In the Watch Window, click the Watch 1 tab.
4. In the Name field, enter `Isr0`.
5. Repeat steps 1 to 3 for `Isr1`, `Isr2`, and `WhatIsActive`.

The global variables `Isr0`, `Isr1` and `Isr2` indicate the number of times the Timer 0, Timer 1 and Timer 2 interrupts have been serviced respectively. The variable `WhatIsActive` contains the interrupt service routine (ISR) that is currently being executed.

6. Run to the `OutputData` breakpoint several times by pressing F5 several times.

The `Isrn` count variables should increase. This indicates that the processor services interrupts when it runs. Note that the variables `Isrn` in the Watch Window may initially be set to 0 or a relatively low number. If the variables do not increase, your interrupts are not configured correctly. You may need to check the `SetupInterrupts()` function to make sure it is correct for your target system.

#### [CCS and Interrupts](#)

7. From the Debug menu, choose Multiple Operation.
8. From the drop-down menu in the Multiple Operation dialog box, select Run (ASM Only). This is the command that is executed.
9. In the Count field, enter 2. Code Composer Studio now executes the program forward through 2 instructions.
10. Click OK to close the dialog box. The processor should now be at the very beginning of Time 3 interrupt `Isr`.
11. From the Debug menu, choose Run.

Now that you have learned how interrupts are handled during stop mode, you are ready for real-time mode.

<file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm>

26/09/2007



### Launching Real-Time Mode

---



REAL - L2

In this lesson, you learn how to enter real-time mode. You also learn how to set refresh rates on Watch windows so that you can see the variables change on time-critical interrupts.

To enter and exit real-time mode, follow these steps:

1. From the Debug menu, choose Real-Time Mode.
2. The status bar at the bottom of the Code Composer Studio window now indicates POLITE REALTIME.
3. From the View menu, choose Real-Time Refresh Options.
4. In the Refresh field, type 400.
5. Check the Global Continuous Refresh checkbox.
6. Click OK to dismiss the dialog box.

You now see the Isr1 and Isr13 variables in the Watch Window incrementing rapidly. The Isr13 variable remains unchanged while the target is halted because Timer 13 is not configured as a time-critical interrupt. Only time-critical interrupts are serviced when the target is halted.

7. From the View menu, choose Real-Time Refresh Options. You can vary how often variables are updated using the Real-Time Refresh Options dialog box. Change it to refresh more frequently and notice how this changes the Watch Window's rate of update.
8. In the Refresh field, change the refresh rate to 100 ms.
9. Observe the Watch Window's refresh rate.
10. Change the refresh rate to 1000 ms.
11. Observe the Watch Window's refresh rate.
12. Now you can set multiple refresh rates. Instead of using global refresh rates for all windows, you can set refresh rates for individual windows by disabling Global Continuous Refresh. From the View menu, choose Real-Time Refresh Options.

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007

### Getting Started With the Code Composer Studio Tutorial

Page 542 sur 548

13. Uncheck the Global Continuous Refresh checkbox. You will no longer see the CPU register or the Watch Windows update.
14. Right-click in the Watch window and select Continuous Refresh. The Watch window now updates, but not the CPU window.
15. From the Debug menu, choose StepIn to step into background code. The windows you have set for continuous refresh continue to update. Notice that Isr0 and Isr1 continue to be serviced.
16. From the Debug menu, choose Real-Time Mode. Selecting a menu item that is checked disables it. The Watch window is no longer continuously refreshed, and the status bar no longer indicates that you are in real-time.  
You can switch to stop mode from POLITE REALTIME when the CPU is halted. In a real-time system, stop mode is used to download the code and initialize the system before interrupts are enabled. After switching back to stop mode, the current PC may still point at code that is executed as part of the time-critical ISR. This is the case if the time-critical ISR was executing when the switch to stop mode occurred.
17. From the Debug menu, choose Run.

Now that you have learned how to enter and exit real-time mode, try debugging a non-time-critical ISR in real-time mode.



### Debugging a Non-Time-Critical ISR In Real-Time Mode

---



REAL - L2

In the previous lesson, using real-time mode enabled you to step through background code while still servicing time-critical ISRs, but not all ISRs are time-critical. It is possible to debug an ISR while in real-time mode by placing a breakpoint within the ISR. In this lesson, you learn how to debug a non-time-critical ISR in real-time mode.

To debug a non-time-critical ISR in real-time mode, follow these steps:

1. From the Debug menu, choose Real-Time Mode to enable real-time mode.
2. From the View menu, choose Real-Time Refresh Options.
3. In the Refresh field, type 400.
4. Check the Global Continuous Refresh checkbox. From the InputData breakpoint in the background code, enable real-time mode, and global continuous refresh at a rate that allows you to observe variables changing without being too distracted. It is recommended that you use 400ms as the refresh rate.
5. Click OK to dismiss the dialog box.

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007

6. From the Debug menu, choose Breakpoints. When polite real-time mode is enabled, you can place a breakpoint within a non-time-critical ISR only after DBGW has been enabled (set to 0). If DBGW is disabled, debug accesses are blocked.
7. Select the Breakpoints tab.
8. In the Breakpoint type field, choose Break at Location.
9. In the Location field, enter Perform1sr3.
10. Click Add to add the breakpoint.
11. Click OK to dismiss the dialog box. You have now moved into non-interruptible code space
12. From the Debug menu, choose Run.
13. You should now be at the first line of Perform1sr3.

A warning message will inform you that the processor has stopped inside non-interruptible code space. This situation occurs because GIE is at 0.

The user can either chose Ignore to continue debugging in real-time mode, or chose OK. When OK is selected, the user can switch to stop mode to debug this non-time-critical interrupt, or continue the debugging in real-time mode by turning on the global interrupt enable but still keeping the interrupt model unaffected by the change. This example is not expecting any other interrupts to happen while inside the non-time-critical interrupt, therefore we should choose the Ignore button. [Note](#)

14. From the Debug menu, choose StepInto. Notice that the CPU registers and the other windows update after each step.  
Repeat Step Into command a few times. [Note](#)
15. From the Debug menu, choose Breakpoints.
16. In the Breakpoint field, select Perform1sr3.
17. Click Delete to remove the breakpoint.
18. Click OK to dismiss the dialog box.
19. From the Debug menu, choose Run.

The processor stops at the OutputData(Result) location in the background again.



file:///C:/Documents and Settings/Pierre-Armand/Local Settings/Temp/~hhBA53.htm

26/09/2007

Getting Started With the Code Composer Studio Tutorial

Page 544 sur 548

### Debugging a Time-Critical ISR in Real-Time Mode



REAL - L2

You may want to debug a portion of a time-critical ISR. Normally, this is done in stop mode, but it is possible for a time-critical ISR to be debugged in real-time mode.

#### [Note](#)

In this tutorial, Perform1sr1() is found in a time-critical interrupt routine. Debug accesses are allowed so that debugging can be done in Polite real-time mode. This lesson shows you how to debug a time-critical ISR in real-time mode.

To debug a time-critical ISR in real-time mode, follow these steps:

1. From the Debug menu, choose Breakpoints.
2. Select the Breakpoints tab.
3. In the Breakpoint type field, choose Break at Location.
4. In the Location field, enter Perform1sr1.
5. Click Add to add the breakpoint.
6. Click OK to dismiss the dialog box. The debugger runs to the breakpoint you just set and then halts.
7. A warning message window also pops up with Ignore and OK buttons. Notice that the Watch window shows that Perform1sr1 and Perform1sr2 are both no longer being serviced. The global interrupt enable GIE is 0 and it has turned off all interrupt except Reset and NMI.
8. Click OK to dismiss the message window.

In this example, the interrupt model is not designed for nesting interrupts. In order to debug the time-critical interrupt in real-time, we need to turn on the global interrupt enable but still keep the interrupt model unaffected. Therefore, before global interrupt is enabled, we need to turn off the IER bits for this interrupt as well as the other time-critical interrupt, and then single-step through Perform1sr1.

1. In the CPU register window, double-click on the IER register and change the value from 0XE003 to 0X2003. This prevents Perform1sr1 from being re-entered, which would block service of the other time-critical interrupt during debugging.
2. In the CPU register window, double-click on the GIE register and change the value from 0 to 1.

file:///C:/Documents and Settings/Pierre-Armand/Local Settings/Temp/~hhBA53.htm

26/09/2007



3. From the Debug menu, choose Stepinto.
4. After finishing debugging this time-critical interrupt, the interrupt setting changes that you made need to be restored to the default. Double-click on the IER register in the CPU register window, change the value back to 0xE003.
5. A warning message will inform you that the processor has stopped inside non-interruptible code space, choose the Ignore button.
6. From the Debug menu, choose Breakpoints.
7. In the Breakpoint field, select Perform1sr1.
8. Click Delete to remove the breakpoint.
9. Click OK to dismiss the dialog box.
10. Double-click on the GIE register to view that the value has changed back to 0x0. Click Done.
11. From the Debug menu, choose Real-time Mode to disable Real-time mode before the next lesson.



### DBGM : Protecting Time-Critical Code from Debug Accesses



REAL - L2

The emulation logic within the C64xx debug architecture has a provision for protecting application code from intrusion by debug accesses made by the emulator. This protection is enabled by setting the Debug Mask (DBGM) bit within the time-critical range of the application.

[For more DBGM information](#)

To set the DBGM bit, follow these steps:

1. Make sure Real-time mode is disabled in the Debug menu. From the Project menu, choose Open.
2. Browse to the `C:\CCStudio_v3.10\Tutorial\emu64xx\realtime` folder and select `dbg.mjt`.
3. Open the file `dbg.m.c` from the Source folder in Project View.
4. Study the function `FIR32`. In this function, the following code section is assumed to be time-critical.

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007



### Getting Started With the Code Composer Studio Tutorial

Page 546 sur 548

```

/* EXAMPLE OF TIME CRITICAL CODE SECTION */
for (i-- ; i >= 0; i--)
{
    sum += ( (int32)x[i] * (int32)coeff[i] ) >> GUARD_BITS;
    x[i+1] = x[i];
}
/* END OF TIME CRITICAL CODE SECTION

```

5. Note that the value of DBGM is set to 1 just prior to the beginning of the *for loop*, and that the value of DBGM is cleared immediately at the end of that loop.
6. From the File menu, choose Load Program.
7. Navigate to the `C:\CCStudio_v3.10\Tutorial\emu64xx\realtime\Debug` folder and select `dbg.m.out`.
8. From the Debug menu, enable Real-Time Mode.
9. The status bar at the bottom of the Code Composer Studio window should indicate POLITE REALTIME.
10. Set a breakpoint at the first line in the `FIR32` function, at line 131. Your line number may vary.
11. From the Debug menu, choose Run.
12. The status of the DBGM bit is located in the Debug Status Register (DBGSTAT). This is a memory mapped register and is located at address 0x01BC0000. The DBGM status is indicated via BIT 26 of the DBGSTAT register. From the View menu, choose Memory and examine the status of DBGSTAT.
13. Observe that the DBGM bit (BIT 26) is clear in the DBGSTAT register.
14. Single step  through the code sequence that sets the DBGM bit. See the comments in the source.
15. Observe that the DBGM bit (BIT 26) is now set.
16. Set a breakpoint at the first instruction past the "for loop" (that is, at `mFreg0_value=0;`).
17. From the Debug menu, choose Run, and run the program to this Breakpoint.
18. Single step  through the code sequence that clears the DBGM bit. See the comments in the source.
19. Observe that the DBGM bit (BIT 26) is now clear in the DBGSTAT register.
20. When DBGM is active, the target is not permitted to halt inside the window of code where DBGM is set. The halt request may be due to a USER halt. Run the target.

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007

21. Apply a halt. Note that the halt will never occur inside the time-critical section ("for loop") of the application.
22. Repeat the sequence in steps 1 and 2 and observe that the target does not halt inside the code section that is guarded by the DBG0M active window.

This concludes the Real-Time Tutorial. For further information on Rude real-time mode and the ABORTI instruction, please see the next section.



### Enabling Rude Real-Time Mode



REAL - L3

**Note:** The information in this section is provided to give you additional information about real-time emulation and is not part of the tutorial.

High priority interrupts, or other sections of code, can be extremely time-critical and the number of cycles taken to execute them must be kept at a minimum or to an exact number. This means debug actions (both execution control and register/memory accesses) may need to be prohibited in some code areas or targeted at a specific machine state. In real-time mode, by default the processor runs in 'polite' mode by absence of privileges, i.e. debug actions will respect the appropriate delaying of the action and not intrude in the debug sensitive windows.

However, debug commands (both execution control and register/memory access) can fail if they are not able to find a window that is not marked debug action sensitive. These cases happen when a real-time ISR sequence has gotten lost, or if the application inadvertently set the DBG0M bit without clearing it. In order to have the debugger gain control, you must change real-time debug from 'polite' to 'rude' mode. In rude real-time mode, the possession of privileges allows a debug action to ignore the debug action sensitive window and be executed successfully without delay.

**To enable Rude real-time mode, perform one of the following:**

- Select Perform a Rude Retry from the display window when a debug command failed, or,
- Select Enable Rude Real-time Mode from Debug menu when real-time is turned on.

When Rude Real-time is enabled, the status bar at the bottom of the main program window displays RUDE REALTIME.

If Rude Real-Time is enabled and you halt the CPU, there is a good chance that the CPU will halt even when debug accesses are blocked, which might be within a time-critical ISR. This prevents the CPU from completing that ISR in the appropriate amount of time, as the CPU cannot do anything until you respond to the breakpoint. To prevent this problem, you must switch back to Polite real-time mode by de-selecting Enable Rude Real-Time Mode.

- From the Debug menu, choose Enable Rude Real-Time Mode to clear the checkmark next to this item. The status bar now reads Polite Real-time Mode.

file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007

Getting Started With the Code Composer Studio Tutorial

Page 548 sur 548



### Aborting Interrupts With the ABORTI Control



REAL - L3

**Note:** The information in this section is provided to give you additional information about real-time emulation and is not part of the tutorial.

Generally, a program uses the B\_IRP or B\_NRP instruction to return from an interrupt. With an unprotected pipeline architecture processor, your application has to be responsible for restoring all the values that have been saved when entering the interrupt. In real-time mode, if a time-critical interrupt is serviced when processor is in debug-halt state, the ISR will complete execution and then return by executing B\_IRP to the stopped code at the same place it was initiated. If a debug event is encountered, the ISR execution stops and multiple code stopped points are created.

As long as the ISR execution is restarted and eventually returned by executing the B\_IRP (same IRP when entering the ISR), debug software can maintain the debug context along the multiple stopped points (debug frame). In some target applications, you may have interrupts that must not be returned from by the B\_IRP instruction to the code where the ISR was initiated. This can cause a problem for the emulation logic because it relies on the correct IRP to keep track of the correct debug context.

The abort interrupt (ABORTI) control capability is provided as a means to indicate that the debug frame is flushed and the debug logic needs to be reset to its default state. To abort an interrupt from its normal operation, the user application needs to write to the memory mapped emulation control register at 0x01bd0014 with a value of 0x60000.

As part of its operation, the ABORTI will reset the execution state to Run, flush the debug frame counter which keeps track of multiple stopped points, and send status notification of a debug synchronization loss. Application should be restarted from a clean state.

The ABORTI control does not modify the DIER registers, the IER registers, the GIE bit, or any analysis registers such as registers used for breakpoints, watchpoints, and data logging.



file://C:\Documents and Settings\Pierre-Armand\Local Settings\Temp\~hhBA53.htm

26/09/2007