Application Report
SPRA381 – April 2002

Example of GEL Usage With File I/O for Code Composer Studio v2.1

Harsh Sabikhi *Code Composer Studio, Applications Engineering*

ABSTRACT

This application report discusses some of the functionality of the General Extension Language (GEL) that is supported by and included with Code Composer Studio™ integrated development environment (IDE) v2.1. Features that will be outlined are automated testing with GEL, workspace customization, and common Code Composer Studio tool usage such as File I/O. In addition, the new GEL application programming interfaces (APIs) for File I/O will be greatly exposed and used in the automation process. First, GEL will be briefly discussed and then two examples will be given to demonstrate the usefulness of the language.

Requirements

- Microsoft® Windows® 98, Windows 2000, Windows NT®, or Windows XP®
- Code Composer Studio Version 2.1

Prerequisites

- Good knowledge of the C programming language
- Good Knowledge of the Code Composer Studio IDE
- Basic knowledge of GEL

Contents

List of Figures

Introduction

Please refer to the Example of GEL Usage With File I/O for Code Composer Studio v2.0 application report (literature number SPRA774) for an introduction to GEL and a background of the procedure used for for Code Composer Studio v2.0.

There have been a number of new GEL APIs added to Code Composer Studio v2.1. However, since this application note describes File I/O using GEL, only the relevant APIs will be discussed.

1 Initializing the Application

In Code Composer Studio v2.0, there was no way of performing File I/O operations in an automated manner without going through the graphical user interface (GUI) in the IDE. If the user creates applications that require the testing of various test vectors, the old method can become very tedious and time-consuming. The File I/O GEL functions for Code Composer Studio v2.1 have corrected this by providing basic File I/O functionality through GEL (as well as indirectly through the APIs), which can then be used in automation scripts. The new functions include GEL_AddInputFile, GEL_AddOutputFile, GEL_RemoveInputFile, and GEL_RemoveOutputFile. Each of these functions can either be specified by program address or by source file name. There are two versions of these GEL functions that allow the user to bind input/output file operations to either a user-specified source file name and line number, or a program memory address. The following is an example of the GEL_AddInputFile and GEL_AddOutputFile in their two forms. The parameters in [ ] are optional.

```
GEL_AddInputFile("srcFileName", lineNumber, "connectFileName", format, "startAddr"
[, "length"] [, page] [, wraparound] [, "condition"])

GEL_AddInputFile(programAddr, "connectFileName", format, "dataAddr" [,"length"]
[, page] [, wraparound] [, "condition"])

GEL_AddOutputFile("srcFileName", lineNumber, "connectFileName", format, "startAddr"
[, "length"] [, page] [, "condition"])

GEL_AddOutputFile(programAddr, "connectFileName", format, "dataAddr" [,"length"]
[, page] [, "condition"])
```

NOTE: This application note makes use of the volume1 project that comes with every Code Composer Studio software package. For this particular application note, the volume.c file had to be altered a little to cater for the particular features of this note. Before proceeding, please replace the volume.c file in the volume1 folder with the one provided in Appendix A.

1.1 Opening a Project Using the Board Start-Up File

First, we will create a custom GEL file that will open the volume1 project, load the COFF file into memory, and connect the Probe Points—all at start-up of Code Composer Studio. This file also contains built-in GEL functions that support performing a ‘file copy’ from GEL. (This will be seen and used later to avoid overwriting user test data results, by copying the generated output data file to an another user-defined data file). The Probe Points are connected using the source line versions of GEL_AddInputFile() and GEL_AddOutputFile(). The copy is performed by the calling the built-in GEL_System() function that executes a DOS command from within the IDE. The output of the DOS command is sent to an output window within the IDE, and only commands that display a text message and require no user interface can be executed. GEL_TextOut() is another built-in function that is used in the volume_test.gel file. This function simply prints a text message to an output window similar to the printf function in the C programming language.

Once you set up your target board through the import configuration dialog box, the GEL file will be called directly from the board setup GEL file.

1. Launch Code Composer Studio
2. First, the developer has to create two data files, one for input and one for output. Call the input file indata1.dat and the output file outdata.dat, which will initially be empty. The contents of indata1.dat are in Appendix B. These files will be used in Section 2 and Section 3.
3. Go to File → New → Source File. Create a GEL file called volume_test.gel by choosing File → Save As. For convenience, save it under the same directory as the volume1 project, which is <install path>\tutorial\<target>\volume1. The source code is provided in Appendix C. Now, exit Code Composer Studio.
4. Open up the CC_Setup menu of Code Composer Studio v2.1. In the import configuration dialog box, choose your specific board and click import, and then close. If there are any previous targets in the system configuration, remove them.
5. In the system configuration, right-click on your board, select properties, and select the Startup GEL File(s) tab. Verify the board start-up GEL file is there and note its location for further use. And now close the board dialog box.
6. Open up Windows Explorer and browse to the location of your board's initial GEL file from Step 4. Open the file using your favorite text editor. In the StartUp function contained within the GEL file, load the volume_test.gel file by typing in **GEL_LoadGel ("<target>\\ tutorial\\<target>\\volume1 \\volume_test.gel")**, save your changes to the same location as the volume_test.gel file as in Step 3, and exit. This will prevent the user from altering the board-specific GEL file.
7. In the file menu of the CC_Setup, choose exit and a dialog box will appear with the question "Save changes to system configuration?", click yes. Now, another dialog box appears with the question "Start Code Composer Studio on exit?", click yes. This will launch Code Composer Studio with the GEL file called at start-up. Notice the features and verify the functionality of the automation. The volume project is loaded as well as the GEL file. Your screen should resemble the screen capture in Figure 1.

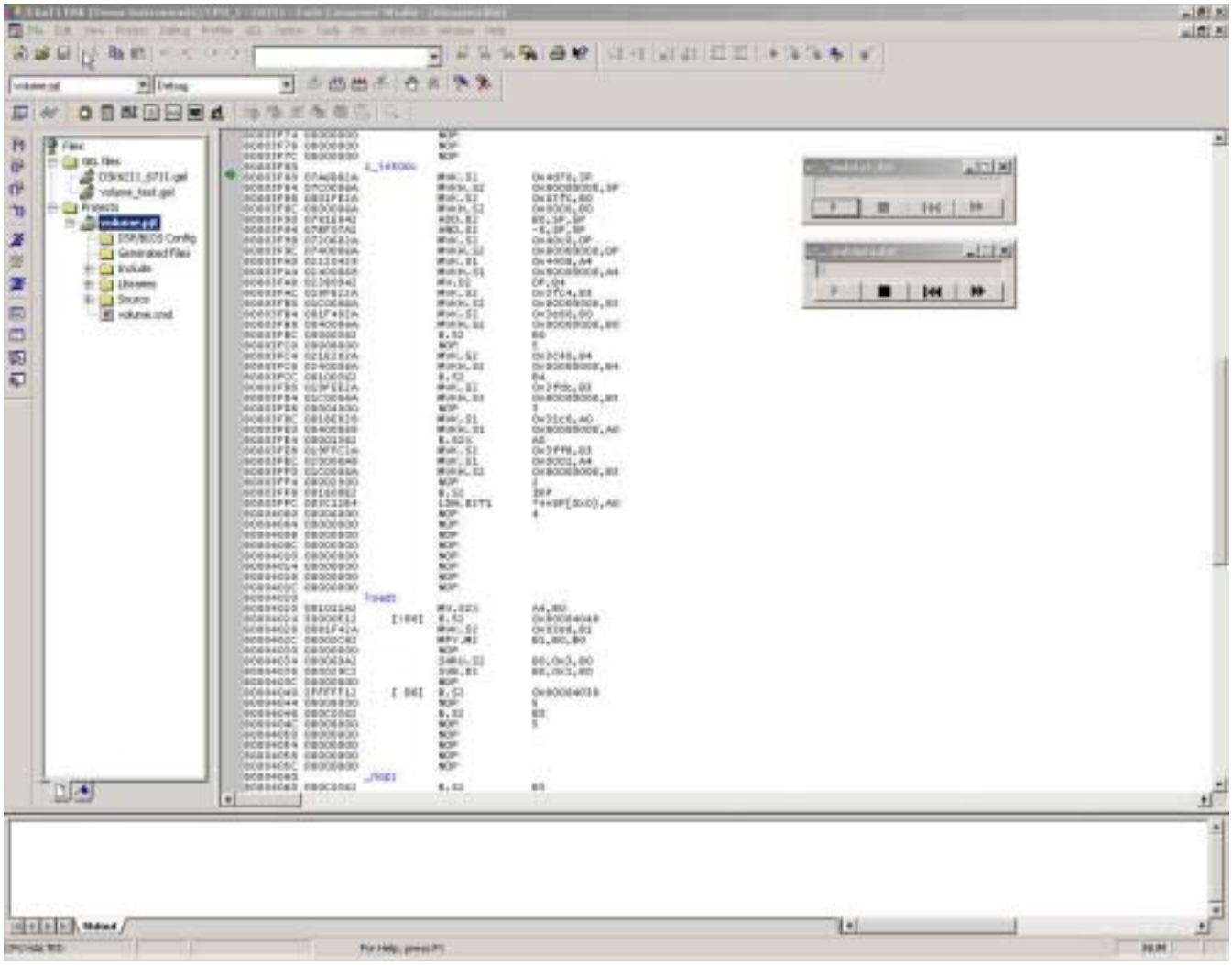

Figure 1. Start-Up View Using Board GEL File

1.2 Opening a Project With a Custom Start-Up File

If your board does not have a start-up file or you would like to create your own, then follow the steps below. Please note if the developer decides not to perform the steps in Section 1.1, Step 2 of Section 1.1 (creating the data files) still has to be completed for this section.

1. Launch Code Composer Studio.
2. Go to File → New → Source File. Create a GEL file called start_volume.gel by choosing File → Save As. For convenience, save it under the same directory as the volume1 project, which is <install path>\tutorial\<target>\volume1. The source code is provided in Appendix D of this document.
3. Repeat Step 2 and create another GEL file called volume_test.gel. The source is provided in Appendix C. Exit Code Composer Studio.

4. Open up the CC_Setup menu of Code Composer Studio v2.1 and close the import configuration dialog box (unless your specific target is not already configured). If your target is already in the system configuration, remove it. This will remove any previous GEL files that were loaded with the board.

5. Drag and drop your board in the system configuration. Click on the processor configuration and add a single processor. Next, click on the Startup GEL File(s) dialog box and click on the browse button. Browse to where the start_volume.gel file is located (refer to Step 2) and click ok and then finish.

6. In the file menu, choose exit and a dialog box will appear with the question “Save changes to system configuration?”, click yes. Now, another dialog box appears with the question “Start Code Composer Studio on exit?”, click yes. This will launch Code Composer Studio with the GEL file called at start-up. Notice the features and verify the functionality of the automation. The GEL files load the volume project as well as copy indata1 into indata.

7. After Code Composer Studio starts, choose File → Load Program, and select the volume.out file. Your screen should resemble the screen capture in Figure 2.

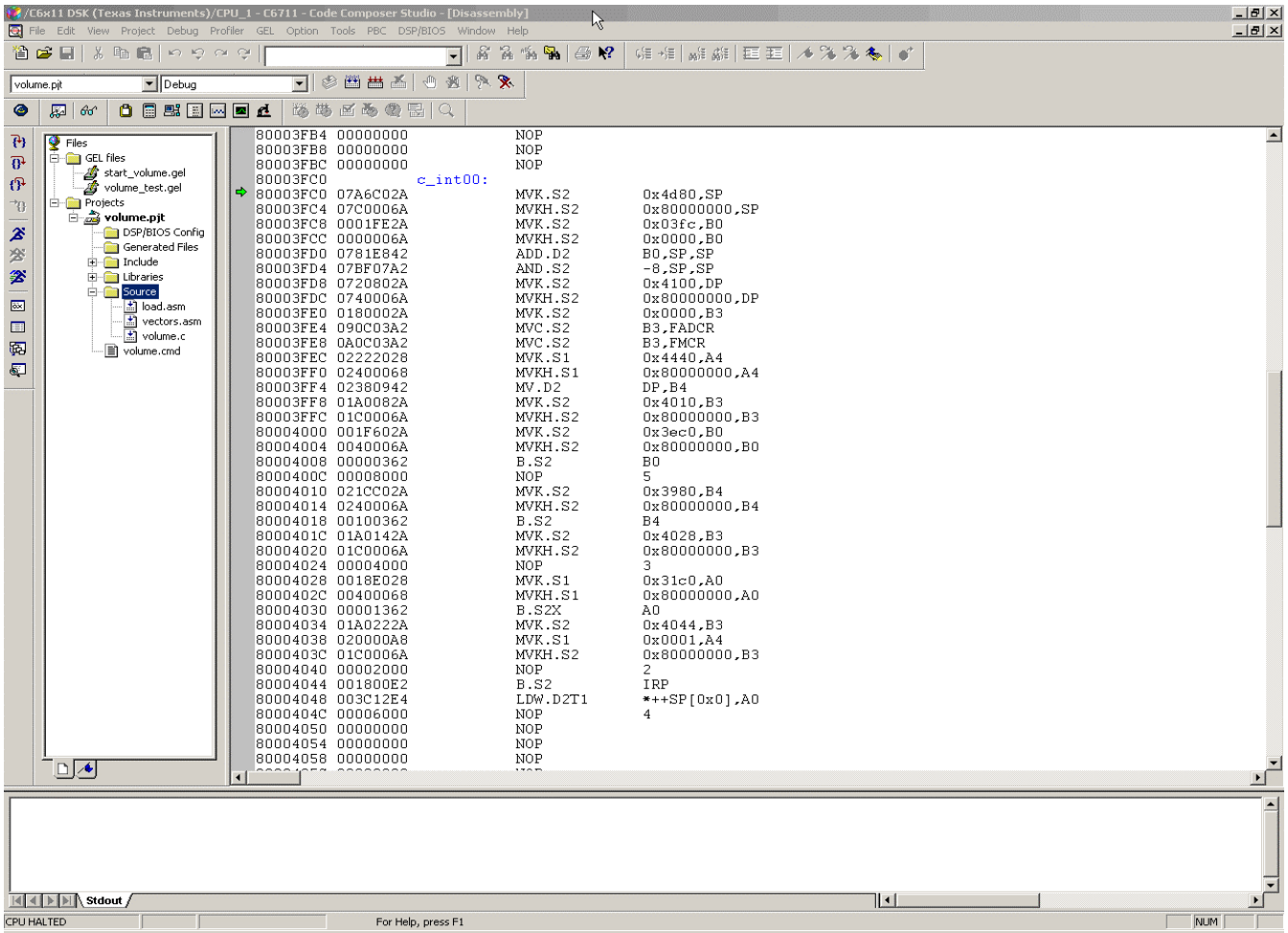

Figure 2. Start-Up View Using Custom GEL File

2 Testing One Vector

2.1 Verifying Files Connected to Probe Points

For the single test vector case that requires only one input and one output vector, no additional code has to be added to the GEL files because the Probe Points are already connected when Code Composer Studio is launched.

1. From Section 1.1 or 1.2, when Code Composer Studio is launched, the volume project is opened. Expand the view and double-click on the volume.c file. Take a second to examine the new source code and read the comments. Please note that in copying and pasting the new code, the line numbers may differ from the ones in Figure 3.

2. To verify the connections, choose File → File I/O and notice that indata1.dat is connected as an input file and outdata.dat is connected as an output file. In addition, if you click on the Probe Point button, you will see a detailed version of the each connection.

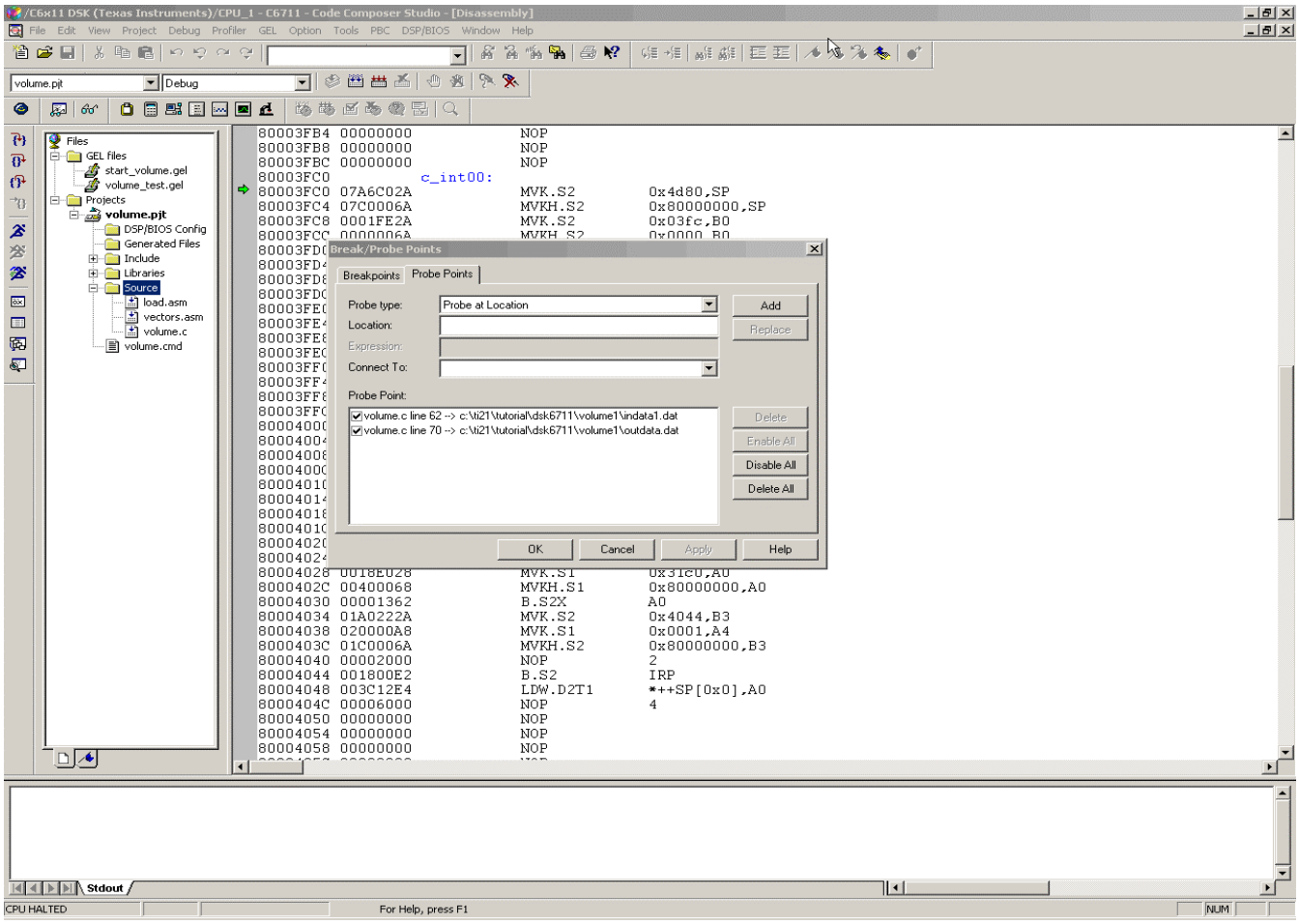

Figure 3. Probe Point Setup Workspace

3. Now that we have successfully added the File I/O functionality to our project, let's run the application and verify the results.

2.2 Running the Application and Verifying the Results

Since this section tests only a single vector, only one output file is copied.

1. Let's make use of GEL now to copy the output file to a user-defined data file. In the volume.c file, toggle a Probe Point on line 74 (puts("Connecting Output File to a user defined Text File\n");). From the debug menu, choose Probe Points and highlight line 74. In the probe type field, select Probe at location if expression is TRUE, and in the expression field, and type in Test_File1() and click Replace. Recall this is a GEL function that copies the output file to a user-defined file for testing purposes. Choose Debug → Run to run the application and watch the output screens to see the execution status. Your screen should resemble the screen capture in Figure 4.

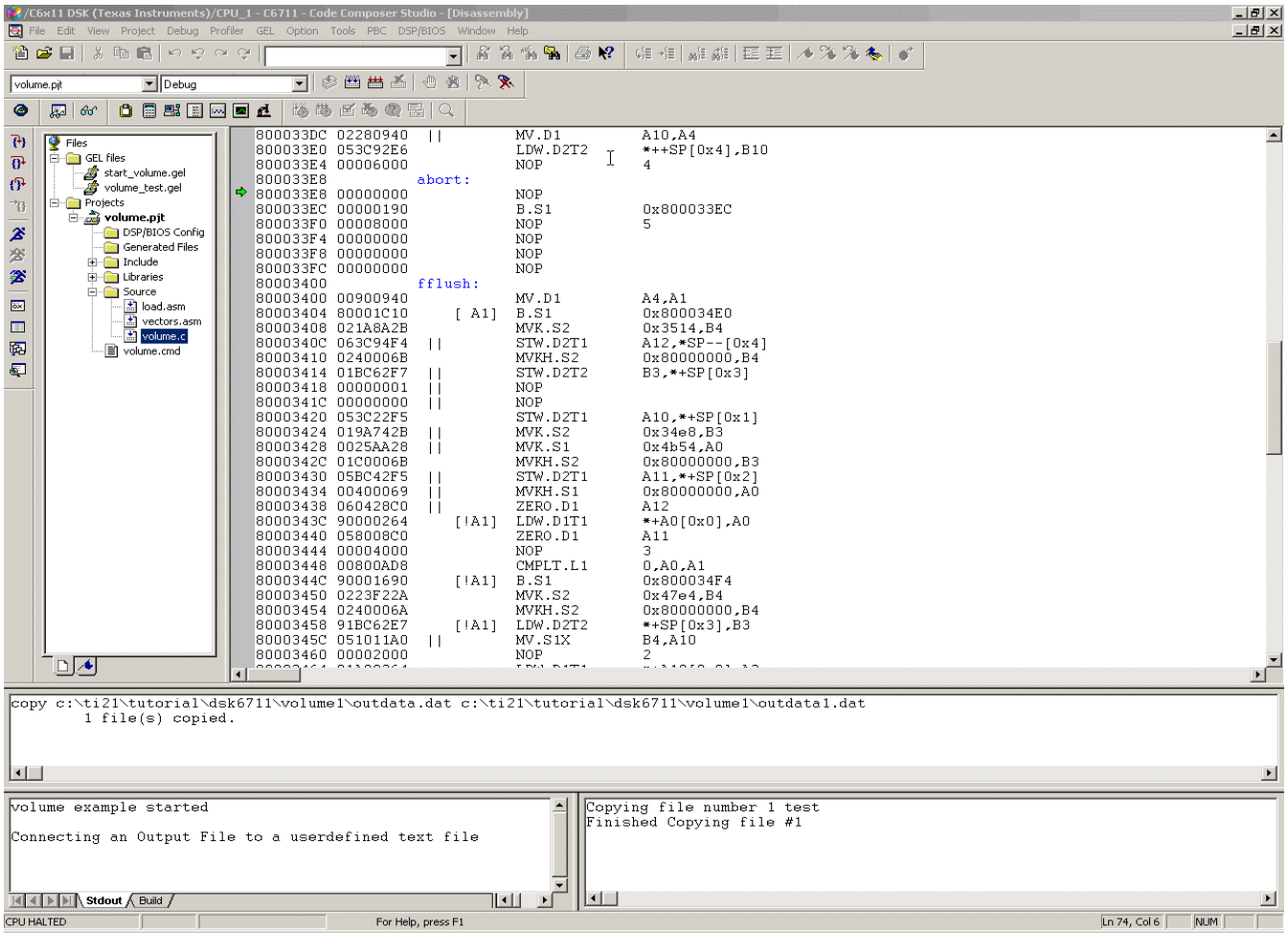

Figure 4. Resultant Output Screen of Single Vector Case

2. Verify the results by checking all of the data files. That is, validate the contents of indata1, outdata, and outdata1. The contents of indata1 should have been probed out to outdata, and finally outdata was copied to outdata1. Since there was no transformation on the input data, the input data file should be equivalent to the output data file.

3 Testing Multiple Vectors

In this section we will start a new workspace to accommodate for the multiple input test vectors. To test multiple vectors using GEL, we cannot have n input and n output Probe Points each connected to a specific GEL function. Rather we have to make use of one entry point and one exit point, since GEL is a scripting language that is asynchronous. The GEL functions that are connected to these Probe Points are dynamic—meaning they depend on a counter. The counter keeps track of the number of test vectors and provides a convenient method of parsing data. At the dataInput() function in our source code, we will connect a single input file that will contain a different input file depending on the counter value. All of the data in these files will be transferred to a single output data file that is connected via the Probe Point at the dataOutput() function.

3.1 Initializing Multiple Test Vector Case

1. Open up Windows Explorer and create three input data files called indata1, indata2, and indata3 that contain the integer values in Appendix B. These data files will be used as input test vectors. If the user has followed Section 2, then indata1 should already exist. Also, we will make use of the output file outdata.dat from Section 2 but delete its contents.

2. First, we will create an input GEL file that, depending on the counter value, will load indata2 or indata3 to be processed. In Code Composer Studio, Select File → New Source file and copy and paste the source code provided in Appendix E. Choose File → Save As and call the file input.gel. Next, we will modify our board GEL file to preload the input.gel and volume_test.gel on start-up of Code Composer Studio.

3. Open up Windows Explorer, browse to the location of your specific board file, and open it using any text editor. We will add GEL syntax in the start-up function to load the two GEL files relevant to this section. Copy and paste the following lines of code in the start-up function.

   GEL_LoadGel(“c:\\ti\\tutorial\\dsk6711\\volume1\\input.gel”);
   GEL_LoadGel(“c:\\ti\\tutorial\\dsk6711\\volume1\\volume_test.gel”);

   Please note, if you performed the steps in Section 2, then the volume_test.gel file should already be in you board file. Save the changes and restart Code Composer Studio with your board file setup (please refer to Section 1.1 for the procedure). Notice the features and functionality of the automation. Your screen should resemble the screen capture in Figure 5.

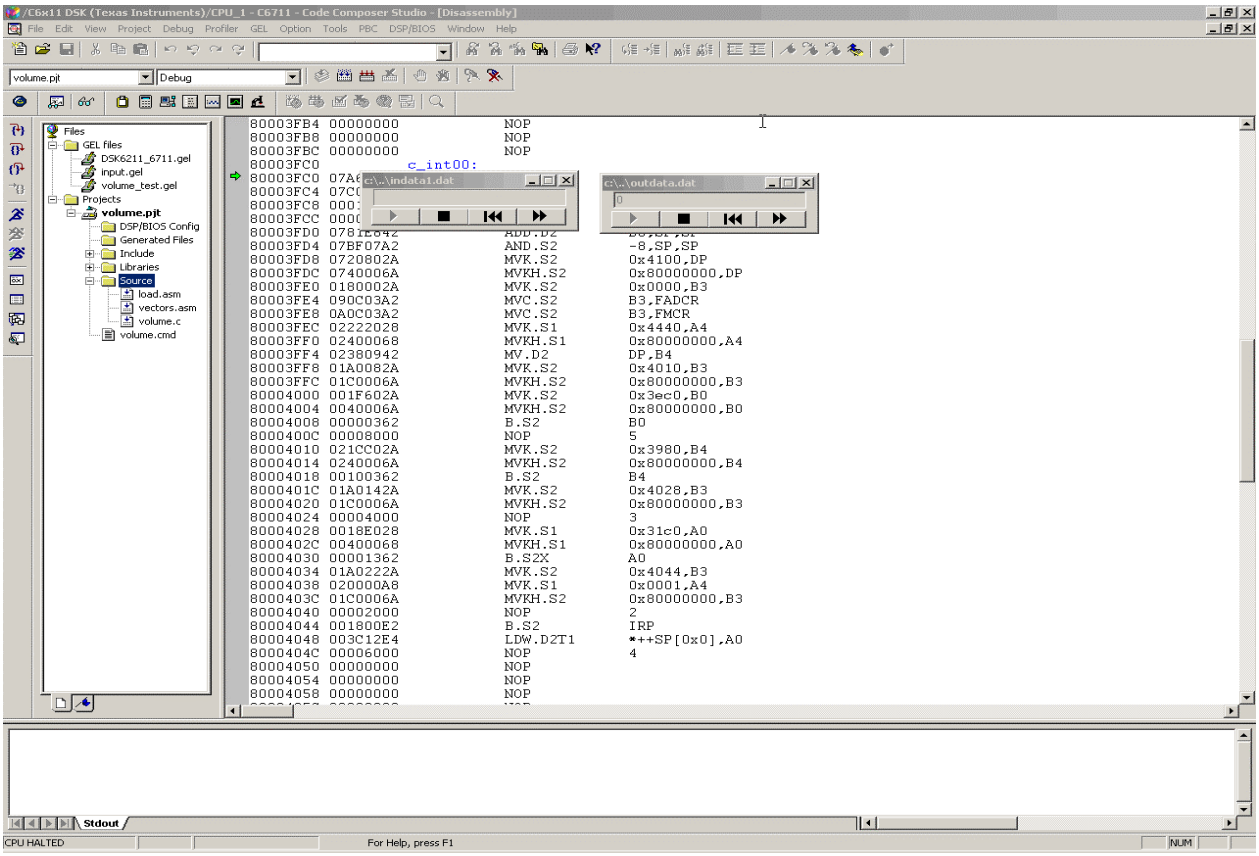

Figure 5. Start-Up View for Multiple Test Vector Case

3.2 Verifying the Probe Points and Running the Application

1. Please make the appropriate changes to the source as specified in the comments on lines 73 and 74. In Code Composer Studio, open up the volume.c file, go to line 73, and uncomment the puts("Connecting a different input file to indata.dat\n");. Next, go to line 74 and comment the puts("Connecting an output file to a user defined data file\n");. Now, rebuild the project.

2. As previously mentioned in Section 2, in the volume_test.gel file under the start-up function, the new GEL APIs are used to connect Probe Points to an input and output file. Initially, indata1.dat is added to line 62 and outdata.dat is connected to line 70.

3. Choose Debug → Probe Point and select line 73. In the Probe type field, choose Probe at Location if expression is TRUE; and in the expression field, type in Input_File(). Click Replace and then ok. The Input_File() function, depending on the counter value, will first remove the old Probe Point, connect a new one, and then restart the program from main.

4. Now, we are ready to run and test the program. Choose Debug → Run. Watch your output screens to check the status of the test. Finally, once all of the data is in outdata.dat, we can make use again of the GEL menu to copy the contents of outdata.dat to some other data file. For illustration purposes, we copy the file to outdata1.dat. Choose

GEL → GEL_Automation → Test_File1. This GEL function was created using the hotmenu keyword and is part of the volume_test.gel file.

5. Verify the results by checking the contents of the data files. The resultant output file outdata.dat is given in Appendix F. After rearranging, the developer's screen should resemble the screen capture in Figure 6.

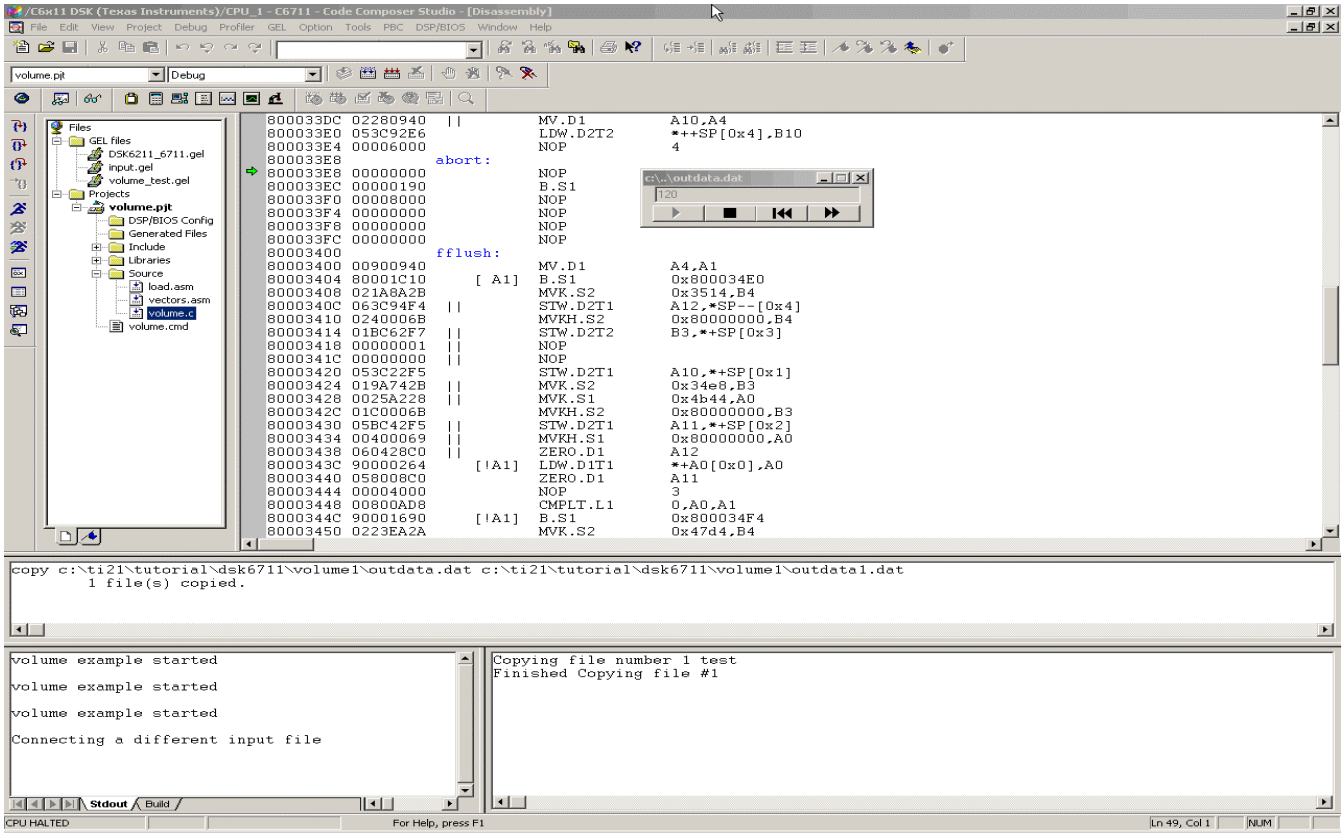

Figure 6. Final Output Screen of Multiple Test Vector Case

4 Callback Function

What makes GEL even more appealing is the fact that it supports a few callback functions. These functions are executed when a particular situation occurs on the target. Currently, there are four callback functions. The first is OnFileLoaded(), which if defined in a loaded GEL file is called after a program is loaded into memory. It takes two integer parameters: one that indicates a success or fail, and the other to indicate if only symbols were loaded. This function has a counterpart OnPreFileLoaded() that requires no input parameters, and if defined in a loaded GEL file, it is called before a program is loaded. Another useful function is OnReset(), which is called when a target processor has been reset. It requires one input parameter to suggest if the call to this function was successful. The last callback function that GEL supports is OnRestart(), which is called when the program is restarted. This function also requires one input parameter to denote if the call to this function was successful.

Conclusion

A question that commonly arises is why we would want to take the time to create these GEL functions? At first, creating these functions is time-consuming but it has long-term benefits. For example, what would happen if we did not load the start_volume.gel file or load our GEL file through the board start-up file? If these files were not loaded, then every time we start Code Composer Studio, the project, along with the associated GEL file(s), would have to be loaded manually. This will, in turn, not only become time-consuming but tedious. GEL is particularly important in automating some common tasks and testing the results of a project. As we have seen, given an input file(s), GEL can buffer out these n files into one output file. In addition, it can copy the resultant output file(s) to any user-defined location for further testing. The developer may want to compare the output with the expected output and generate a difference table.

Throughout this report, many built-in GEL functions were used. Only those functions new for Code Composer Studio v2.1 were explained in detail. For all other functions related to this application note, please refer to the Example of GEL Usage With File I/O for Code Composer Studio v2.0 application report (literature number SPRA774). For more information on other functions, please refer to the online help or to the contents menu under help in Code Composer Studio.

Although the application itself is quite general, the developer should definitely extend the examples provided to fit their specific needs. Furthermore, for the multiple test vector case, only three test vectors were used. This can be extended to hundreds of test vector cases by making little changes to the GEL files.

Appendix A. Source Code

```
/*
 *  Copyright 2001 by Texas Instruments Incorporated.
 *  All rights reserved. Property of Texas Instruments Incorporated.
 *  Restricted rights to use, duplicate or disclose this code are
 *  granted through contract.
 *  U.S. Patent Nos. 5,283,900  5,392,448
 */
/* "@(#) DSP/BIOS 4.51.0 05-23-01 (barracuda-i10)" */
/************************************************************************/
/*                                                                      */
/*     V O L U M E . C                                                  */
/*                                                                      */
/*     Audio gain processing in a main loop                             */
/*                                                                      */
/************************************************************************/

#include <stdio.h>

#include "volume.h"

/* Global declarations */
int inp_buffer[BUFSIZE];                /* processing data buffers */
int out_buffer[BUFSIZE];

int gain = MINGAIN;                     /* volume control variable */
unsigned int processingLoad = BASELOAD;  /* processing routine load value */

struct PARMS str =
{
    2934,
    9432,
    213,
    9432,
    &str
};

/* Functions */
extern void load(unsigned int loadValue);

static int processing(int *input, int *output);
static void dataInput(void);           //Split the old dataIO function into two functions
static void dataOutput(void);

/*
 * ======== main ========
 */

void main()
{
 int iterator = 0;                     //internal counter to loop around each data file
 int *input = &inp_buffer[0];
 int *output = &out_buffer[0];

   puts("volume example started\n");

   while(iterator<10)//Since for this specific example there are 10 data points in each file
   {                 //iterator counts from 0 to 9.  The developer can modify this for any
```

```
                    //number of data points
       /*
        *  Read input data using a probe-point connected to a host file.
        *  Write output data to a graph connected through a probe-point.
        */
        dataInput();                    //This function is used for input data

        #ifdef FILEIO
        puts("begin processing");
        #endif

        /* apply gain */
        processing(input, output);
        dataOutput();                   //This function is used for output data
        iterator++;
   }
  puts("Connecting a different input file to indata.dat\n");//Used in section 3 only. Please
                                                           //comment it out for section 2
  puts("Connecting an Output File to a user defined data file\n");//Used in section 2 only
}                                                        //Please comment it out for section 3

/*
 *  ======== processing ========
 *
 * FUNCTION: apply signal processing transform to input signal.
 *
 * PARAMETERS: address of input and output buffers.
 *
 * RETURN VALUE: TRUE.
 */
static int processing(int *input, int *output)
{
    int size = BUFSIZE;

    while(size--){
        *output++ = *input++ * gain;
    }

    /* additional processing load */
    load(processingLoad);

    return(TRUE);
}

/*
 *  ======== dataInput ========
 *
 * FUNCTION: read input signal and write processed output signal.
 *
 * PARAMETERS: none.
 *
 * RETURN VALUE: none.
 */
static void dataInput()
{
    /* do data I/O */

    return;
}
```

```
static void dataOutput()
{
  /* do data I/O */
  return;
}
```

Appendix B. Input Data Files

```
//File 1
//Data for Indata1

1651 1 0 0 0    //Data file header with magic number 1651.  For more information on file headers
9               //please refer to online help
1
5
8
9
7
1
2
5
4

//File 2
//Data for Indata2
1651 1 0 0 0
9
1
5
7
9
6
7
5
6
4

//File 3
//Data for Indata3
1651 1 0 0 0
4
1
6
6
1
6
5
8
5
9
```

Appendix C. Volume_test.gel

```
StartUp()
{
   GEL_ProjectLoad("C:\\ti21\\tutorial\\dsk6711\\volume1\\volume.pjt");
   GEL_AddInputFile("volume.c",62,"c:\\ti21\\tutorial\\dsk6711\\volume1\\indata1.dat",2,
   "inp_buffer","1",1,1,);
   GEL_AddOutputFile("volume.c",70,"c:\\ti21\\tutorial\\dsk6711\\volume1\\outdata.dat",2,
   "out_buffer","1",1,0,);
}

//This has been added just as an alternative to loading the files on startup
menuitem "GEL_Automation"
hotmenu Add_Files()
{
   GEL_AddInputFile("volume.c",62,"c:\\ti21\\tutorial\\dsk6711\\volume1\\indata1.dat",2,
   "inp_buffer","1",1,1,);
   GEL_AddOutputFile("volume.c",70,"c:\\ti21\\tutorial\\dsk6711\\volume1\\outdata.dat",2,
   "out_buffer","1",1,0,);

}

hotmenu Test_File1()
{
   GEL_TextOut("Copying file number 1 test\n");
   GEL_System("copy c:\\ti21\\tutorial\\dsk6711\\volume1\\outdata.dat
   c:\\ti21\\tutorial\\dsk6711\\volume1\\outdata1.dat");
   GEL_TextOut("Finished Copying file #1\n");
}
```

Appendix D. start_volume.gel

```
//This GEL File loads the volume1 project on start up as well as the volume_test.gel file
//This piece of code is used only for section 1.2 and is intended for those developers who
//wish to create their own custom startup file rather than using the board specific one

//Please note this code was written specifically for the C6711 dsk.
//The developer has to modify the target to work for their board

StartUp()
//anything defined in this function will be executed once the GEL file is loaded
{

 GEL_ProjectLoad("C:\\ti21\\tutorial\\dsk6711\\volume1\\volume.pjt");//Load the volume.pjt
                                                                    //project
 GEL_ProjectRebuildAll();          //Rebuilds all of the files associated with the project

 GEL_LoadGel("C:\\ti21\\tutorial\\dsk6711\\volume1\\volume_test.gel");
}
```

Appendix E. Input.gel file

```
//When this GEL file is first loaded into program memory, the startup function is called and
//a 0 is assigned to a free memory location on the dsk6711
//Note: 0xFE00 is a free memory location only on the dsk6711 (and in this particular
//application), please modify the code for your specific target


StartUp()
{
   *((int*)(0xFE00)) = 0;          // set "counter" to 0 at startup
}

Input_File()
{
   int counter=*((int*)(0xFE00));  //update "counter" with previous value

   if(counter==0)            //when this function is first called, the value of counter is
   {                         //zero, thus the if loop is invoked and indata2 is ready
     GEL_RemoveInputFile("volume.c",62,"C:\\ti21\\tutorial\\dsk6711\\volume1\\indata1.dat");
     GEL_AddInputFile("volume.c",62,"c:\\ti21\\tutorial\\dsk6711\\volume1\\indata2.dat",2,
     "inp_buffer","1",1,1,);
     GEL_Restart();         //this function restarts the target application
     GEL_Go(main);          //from main
     GEL_Run();             //and run the program on the target

   }

   if(counter==1)          //after executing the first if loop, the counter is incremented
   {                       //and indata3 is now ready
     GEL_RemoveInputFile("volume.c",62,"C:\\ti21\\tutorial\\dsk6711\\volume1\\indata2.dat");
     GEL_AddInputFile("volume.c",62,"c:\\ti21\\tutorial\\dsk6711\\volume1\\indata3.dat",2,
     "inp_buffer","1",1,1,);
     GEL_Restart();       //this function restarts the target application
     GEL_Go(main);        //from main
     GEL_Run();           //and run the program on the target

   }
                          //The user can define more similar if loops to accommodate for
                          //the number of files to test

   counter++;
   *((int*)(0xFE00)) = counter;       //update memory location with counter value
   return (1);
}
```

Appendix F. Outdata.dat

1651 1 0 0 0
0x00000004
0x00000001
0x00000005
0x00000008
0x00000009
0x00000007
0x00000001
0x00000002
0x00000005
0x00000004
0x00000009
0x00000001
0x00000005
0x00000007
0x00000009
0x00000006
0x00000007
0x00000005
0x00000006
0x00000004
0x00000009
0x00000001
0x00000006
0x00000006
0x00000001
0x00000006
0x00000005
0x00000008
0x00000005
0x00000009

IMPORTANT NOTICE

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

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

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

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

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

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

Mailing Address:

Texas Instruments
Post Office Box 655303
Dallas, Texas 75265