Application Report
SPRA774 - August 2001

Example of GEL Usage with File I/O for Code Composer Studio v2.0

Harsh Sabikhi *Code Composer Studio, Applications Engineering*

ABSTRACT

This application report discusses some of the general attributes of the General Extension Language (GEL) that is supported by and included with Code Composer Studio™ (CCStudio) Integrated Development Environment (IDE). Certain features that will be outlined are automated testing with GEL, workspace customization, and common CCStudio tool usage such as File I/O. GEL will be briefly discussed and then two examples will be given to demonstrate the usefulness of the language.

Requirements:

- Microsoft Windows 98, 2000 or NT
- Code Composer Studio IDE Version 2.0

Prerequisites:

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

Contents

List of Figures

1 Introduction

At times, some applications require tools to follow certain guidelines. When working on large projects, it is advantageous to have certain recurring tasks automated. General Extension Language (GEL) is an interpretive language similar to C that lets you create functions to extend Code Composer Studio™ (CCStudio) IDE capabilities. GEL is primarily used for creating custom functions to do specific tasks. The developer can create GEL functions using the GEL syntax and then load them into CCStudio. With GEL, you can access actual/simulated target memory locations and add options to CCStudio GEL menu. You can also add GEL functions in the Watch Window so they execute at every breakpoint/Probe Point. In addition, some of the C language syntax such as for, while, and if and else–if loops are all valid in GEL. GEL is particularly useful for automated testing and user-workspace customization. A GEL file can contain many GEL function definitions. This scripting language differs from C in the sense that these functions do not identify any return type or need any header information to define the types of parameters they require. This information is obtained automatically from the data value. GEL is asynchronous which means that when GEL functions are called, they do not wait for any other function to complete executing. This language is very flexible that makes it very easy to program and allows the developer to create a wide variety of applications ranging in complexity.

This application report makes use of built-in GEL functions, custom functions, and keywords to create a customized application. Built-in functions are discussed throughout the application report and callback functions are described in section 5. In addition, there is a special function called StartUp() that can be called only once per GEL file. Anything declared in this function is executed first when the file is loaded. Keywords are used to add GEL functions to the GEL menu bar of CCStudio. The menu item keyword is used to create a new drop-down list of items in the GEL menu. You can then use the keywords hotmenu, dialog, or slider to add new items in the most recent drop-down list. When you select the user-defined menu item (under the GEL menu), a dialog box or slider object appears. The hotmenu keyword adds a GEL function to the GEL menu and when selected, it immediately executes that specific function. The dialog keyword creates a GUI dialog window for parameter entry. Once the appropriate parameters are entered, press the Execute button to call the function. The slider keyword creates an adjustable controller

that can vary the value of a single parameter. In addition to all of these excellent features, CCStudio also comes integrated with a GEL tool bar so the developer can directly make function calls from within the IDE. This tool bar consists of an expression field and an Execute button. To invoke any GEL statement or user-defined function, enter the appropriate function call in this field box and press Execute to evaluate the expression. The expression dialog box maintains a history of the most recently invoked GEL statements/user-defined functions; you may select any of these using the scroll buttons.

There is no limit to the number of GEL functions that can be defined. However, the number of GEL functions that appear on the GEL menu may be limited by the size of your screen and the monitor resolution.

This application report was done on a dsk6711; therefore, all of the GEL code and files have been referenced based upon this particular board. However, with a few minor changes to location and/or target, the developer can use this application report for any target including simulators.

2 Initializing the Application

GEL can be used during startup of CCStudio to run some preliminary tasks such as loading a project, opening a GEL file, or executing code on the target. The developer can customize the IDE using some GEL functions. Customizing the IDE is very useful for developers who are working on large projects and every time they start CCStudio, a general set of windows or tasks have to be opened or executed. A practical example is given to illustrate these features.

2.1 Opening a Project Using the Board Startup File

We create a custom GEL file that at startup of CCStudio opens the volume1 project, builds it, and copies the contents of indata1.dat into indata.dat. This file also contains custom GEL functions that copy an output file to a user-defined data file. This is done with the calling of the built-in GEL_System() function that executes a DOS command 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 are executed. The GEL_TextOut() is another built-in function that is used in the volume_test.gel file. This function prints a text message to an output window. Once you set up your target board through the Import Configuration dialog box, the GEL file is called directly from the board setup GEL file.

1. Launch CCStudio.
2. Create three data files: one for input, one for output, and one for loading the input data. Name the input file indata.dat, which initially is blank. Name the output file outdata.dat, which also is blank. And finally, create an input file named indata1.dat, whose contents are copied into indata.dat. The content of indata1.dat is in Appendix D. These files are used in both sections 3 and 4.
3. Go to File ⇒ New ⇒ Source File. Create a GEL file named volume_test.gel and go to File ⇒ Save As. For convenience, save the file in the same directory as the volume1 project, that is, <*install path*>\tutorial\<*target*>\volume1. The source code for volume_test.gel is in Appendix B.
4. Exit CCStudio.
5. Open the CCStudio Setup and in the Import Configuration dialog box, select your specific board, click Import, and click Close. If there were any previous targets in the System Configuration, remove them.

6. In the System Configuration, right click on your board, select Properties, and select the Startup GEL File(s) tab. Verify the board startup GEL file is there and note its location for future use. Close the Board Properties dialog box.

7. Open Windows Explorer and browse to the location of your board's initial GEL file from step 3. Open the file using any text editor. In the StartUp function within the GEL file, type:
`GEL_LoadGel ("c:\\ti\\tutorial\\dsk6711\\volume1\\volume_test.gel")`
save your changes and exit.

8. In the CCStudio Setup, go to File ⇒ Exit. A dialog box appears asking "Save changes to system configuration?". Click Yes. Another dialog box appears asking "Start Code Composer Studio on exit?". Click Yes. This launches CCStudio with the GEL file called at startup. Notice the features and verify the functionality of the automation. The volume project and GEL file are loaded. Your screen should look similar to Figure 1.

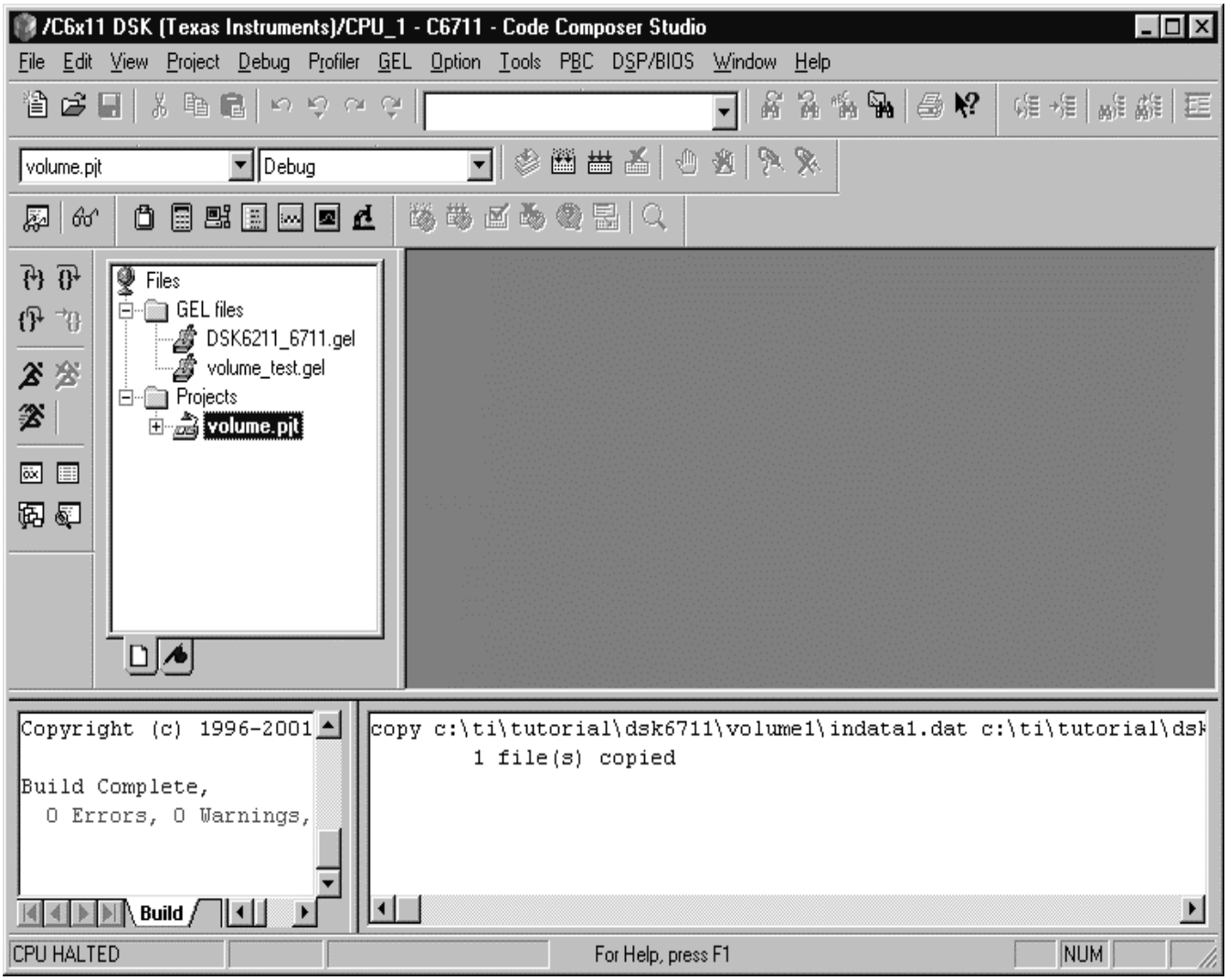

Figure 1. Startup View Using Board GEL File

2.2 Opening a Project With a Custom Startup File

If your board does not have a startup file or you would like to create your own, follow the steps below. If the developer decides not to perform section 2.1, step 2 of section 2.1 still must be completed for this section.

1. Launch CCStudio.
2. Go to File ⇒ New ⇒ Source File. Create a GEL file named start_volume.gel and go to File ⇒ Save As. For convenience, save the file in the same directory as the volume1 project, that is, <*install path*>\tutorial\<*target*>\volume1. The source code for start_volume.gel is in Appendix A.
3. Go to File ⇒ New ⇒ Source File. Create a GEL file named volume_test.gel and go to File ⇒ Save As. For convenience, save the file in the same directory as the volume1 project, that is, <*install path*>\tutorial\<*target*>\volume1. The source code for volume_test.gel is in Appendix B.
4. Exit CCStudio.
5. Open the CCStudio Setup 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 removes any previous GEL files that were loaded with the board.
6. Drag and drop your board in the System Configuration. Select the Processor Configuration tab and click Add Single processor.
7. Select the Startup GEL File(s) tab and click on the browse button. Browse to where the start_volume.gel file is located (refer to step 2), click OK, and click Finish.
8. Go to File ⇒ Exit. A dialog box appears asking “Save changes to system configuration?”. Click Yes. Another dialog box appears asking “Start Code Composer Studio on exit?”. Click Yes. This launches CCStudio with the GEL file called at startup. Notice the features and verify the functionality of the automation. The GEL files load the volume project and copy indata1 into indata. Your screen should look similar to Figure 2.

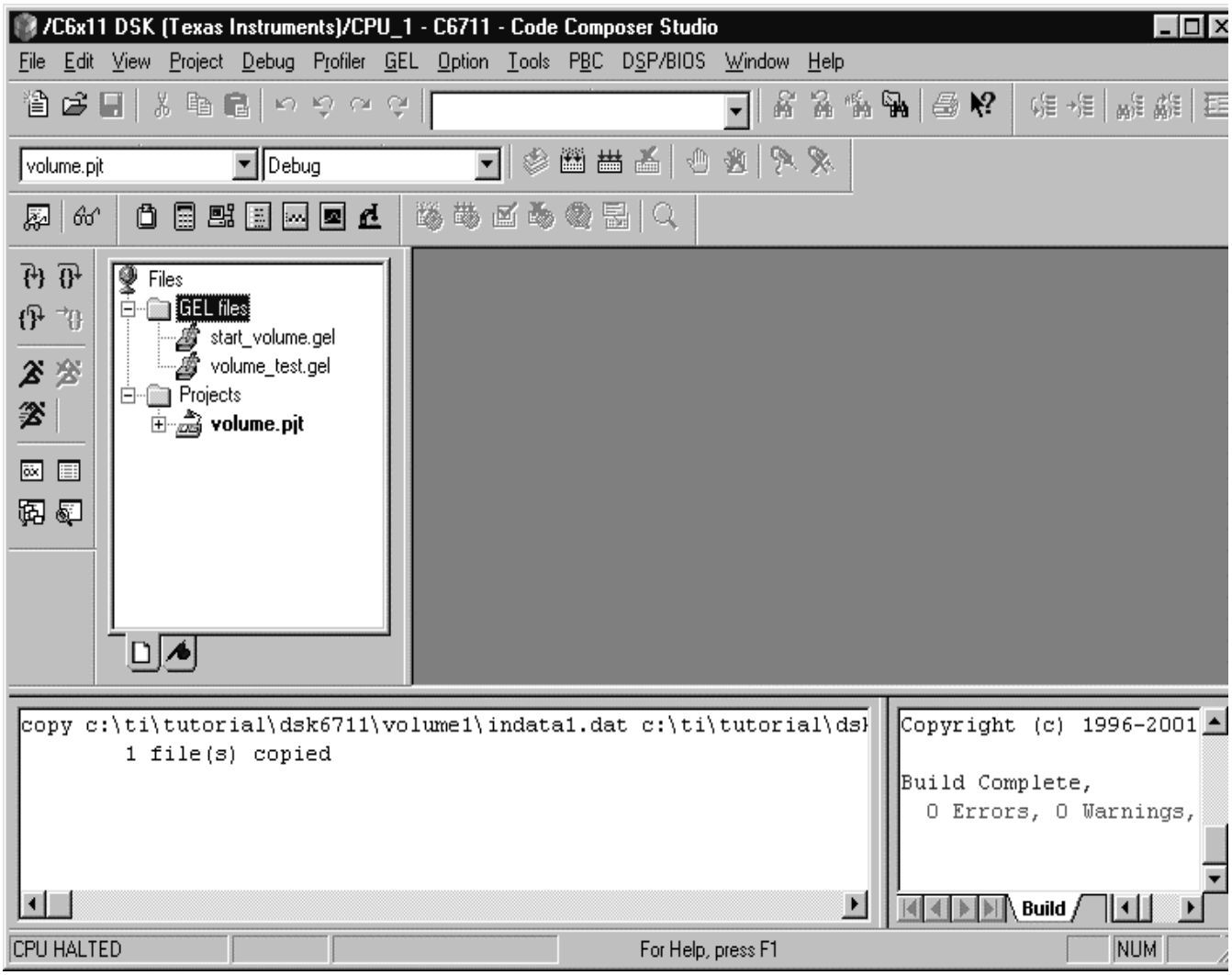

Figure 2. Startup View Using Custom GEL File

3 Testing Single Vector

3.1 Saving a Workspace With Files Connected to Probe Points

Now we make use of the File I/O tool in CCStudio to inject sample data points from an input file to an output file. This has to be done manually only once, and then it can be loaded automatically on startup of CCStudio. This section makes use of the volume1 project, except with a few minor changes to the source code of volume.c.

1. From section 2.1 or 2.2, when CCStudio is launched, the volume project is opened. Expand the view and double click on the volume.c file. Copy and paste the new source code in Appendix C and save the changes.
2. Review the new source code and read the comments. Note in copying and pasting the new code into CCStudio, the line numbers may differ from those mentioned in the following steps.
3. Before connecting Probe Points to the source file, we have to load the program into memory. In CCStudio, go to File ⇒ Load Program and browse to your volume.out file. Go to line 62 (dataInput();) and right click to select toggle Probe Point.
4. Go to line 70 (dataOutput();) and right click to select toggle Probe Point.
5. Go to File ⇒ File I/O. In the File I/O dialog box, select the File Input tab and click Add File. Browse to <*install path*>\tutorial\<*target*>\volume1 and select indata.dat. In the Address field, type inp_buffer; in the Length field, type 1; and enable Wrap Around.
6. Now we connect this file to a Probe Point. In the File I/O dialog box, click Add Probe Point and select line 62. In the Connect To field, select FILE IN: C:\…\indata.dat, click Replace, and click OK.
7. Select the File Output tab, click Add File. Name the output file outdata.dat and click OK. In the Address field, type out_buffer; in the Length field, type 1.
8. Click Add Probe Point and select line 70. In the Connect To field, select FILE OUT: C:\…\outdata.dat, click Replace, click OK, and click OK to exit the File I/O dialog box.
9. Go to Debug ⇒ Probe Points. Your screen should look similar to Figure 3.
10. Now that we have successfully added the File I/O functionality to our project, we save the workspace file. Go to File ⇒ Workspace ⇒ Save Workspace As. Name the workspace file volume_run.wks. For convenience, save the file in the same directory as the volume1 project.
11. Exit CCStudio.

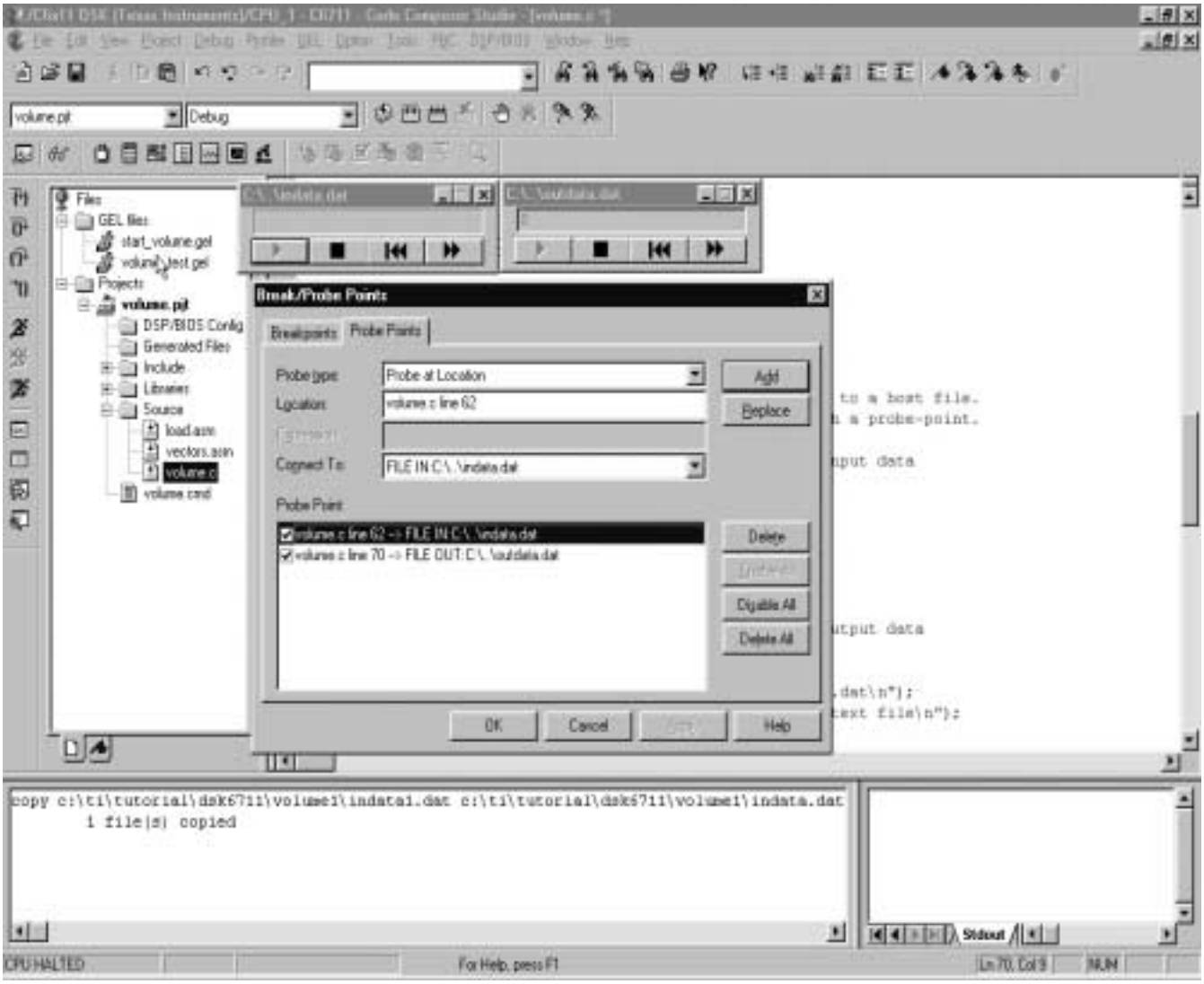

Figure 3. Probe Point Setup Workspace

3.2 Running the Application and Verifying the Results

Now we open the workspace that we created in section 3.1 on startup and then run the application and verify the results.

1. In Windows, right click on the CCStudio icon and go to Properties.
2. In the CCS Properties dialog box, select the Shortcut tab and verify that the target field contains the path name and file name of the CCStudio executable. For example: c:\ti\cc\bin\cc_app.exe.
3. At the end of this path name, add the name of your workspace file. For example: c:\ti\cc\bin\cc_app.exe volume_run.wks.
4. In the Start in field, type the path to your workspace file. For example: c:\ti\tutorial\dsk6711\volume1.
5. Click Apply and close the CCS Properties dialog box.

6. Start CCStudio to verify the process.
7. Before continuing, the program has to be loaded. Go to File ⇒ Load Program. Now we use GEL 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");).
8. Go to Debug ⇒ Probe Points and select line 74. In the Probe type field, select Probe at Location if expression is TRUE; in the Expression field, type Test_File1(). Recall this is a GEL function that copies the output file to a user-defined file for testing purposes.
9. Go to Debug ⇒ Run to run the application and watch the output screens to see the execution status. Your screen should look similar to Figure 4.

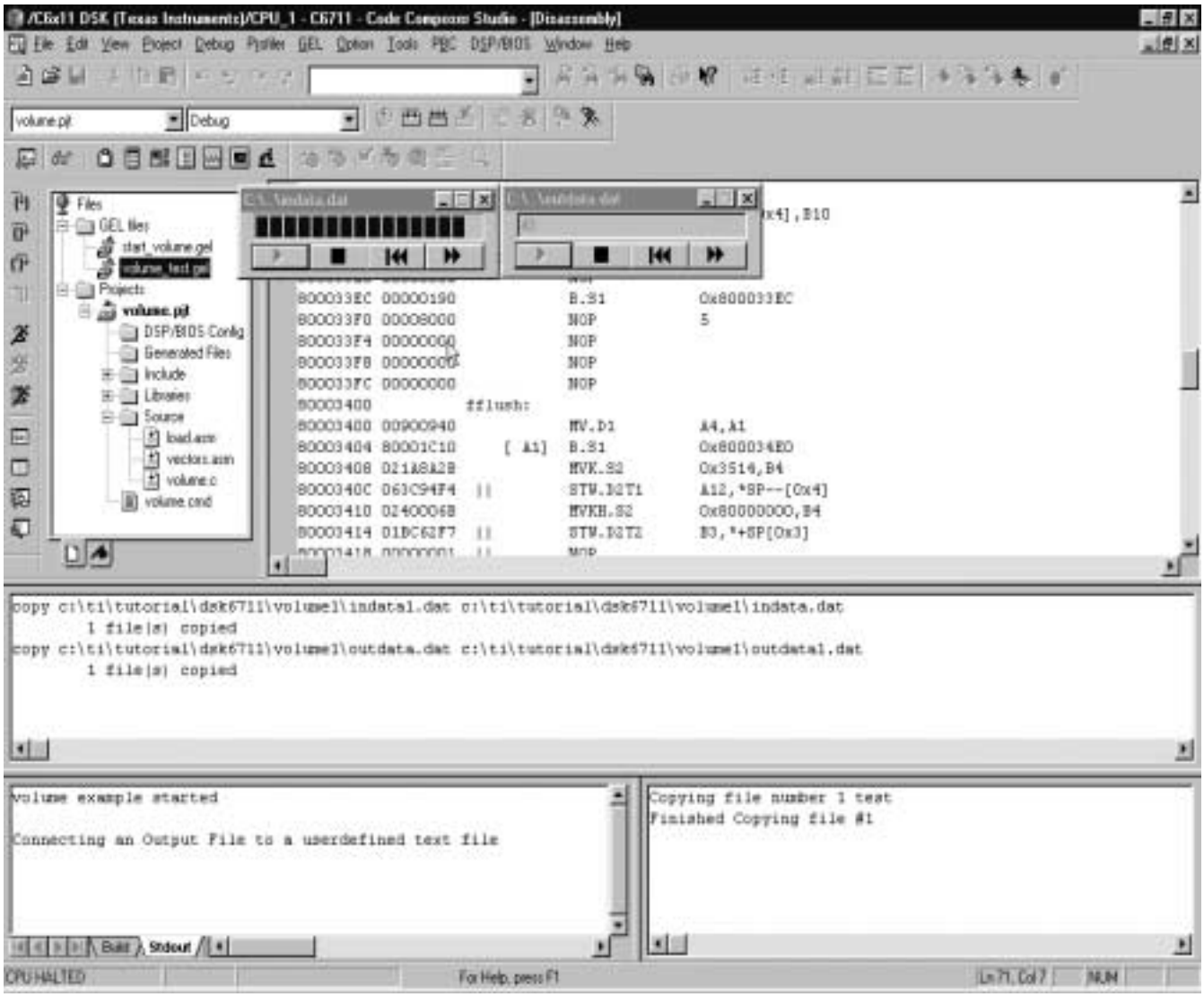

Figure 4. Output Screen of Single Vector Case

10. Verify the results by checking all of the data files. That is, validate the contents of indata1, indata, outdata, and outdata1. Indata1 should have been copied into indata, which was probed 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.

For those users that did section 3 and now are proceeding to section 4, the workspace file that is loaded automatically at startup has to be removed. Remove the volume_run.wks from the CC_apps icon before starting section 4.

4 Testing Multiple Vectors

In this section we 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 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 connect a single input file that contains a different input file depending on the counter value. All of the data in these files is transferred to a single output data file that is connected via the Probe Point at the dataOutput() function.

4.1 Initializing Multiple Test Vector Case

1. Open Windows Explorer and create three input data files: indata1, indata2, and indata3. These input data files contain the integer values in Appendix D. The data files that are used as test vectors are copied into indata.dat, which initially is blank. If the user has followed section 3, then indata1 should already exist. Also, we make use of the output file outdata.dat from section 3, but delete its contents.
2. Create an input gel file that copies indata1 into indata.dat on startup and depending on the counter value loads indata2 or indata3 into indata.dat. In CCStudio, go to File ⇒ New ⇒ Source File. Create a GEL file named input.gel and go to File ⇒ Save As. Copy and paste the source code for input.gel from Appendix E.
3. Modify the board GEL file to preload the input.gel and volume_test.gel on startup of CCStudio. The volume_test.gel file has to be modified for this section. In the StartUp function, comment out the GEL_System command line since this command is replicated in the StartUp function of the input.gel file.
4. Open Windows Explorer and browse to the location of your specific board file. Open the file using any text editor. We add GEL syntax in the startup function to load the two gel files relevant to this section. Copy and paste the following lines of code in the startup function:

```
GEL_LoadGel("c:\\ti\\tutorial\\dsk6711\\volume1\\input.gel");
GEL_LoadGel("c:\\ti\\tutorial\\dsk6711\\volume1\\volume_test.gel");
```

5. If you did the previous sections, the volume_test.gel file should already be in you board file. Save the changes and restart CCStudio with your board file setup (refer to section 2.1 for the procedure). Notice the features and functionality of the automation. Your screen should look similar to Figure 5.

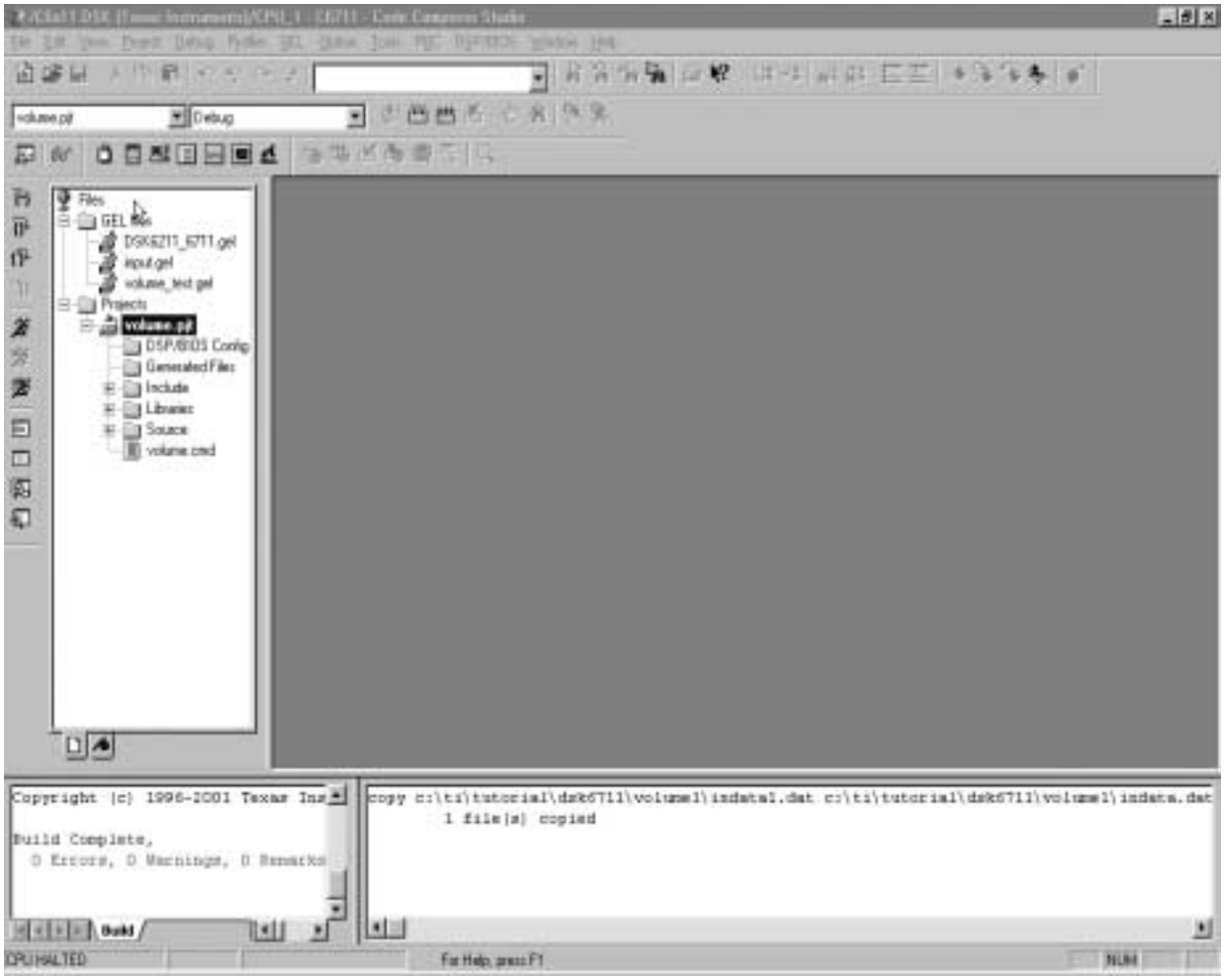

Figure 5. Startup View for Multiple Test Vector Case

4.2 Setting Up Probe Points and Running the Application

1. Before connecting Probe Points to the source file, we have to load the program into memory and make the appropriate changes to the source as specified in the comments on lines 73 and 74. In CCStudio, go to File ⇒ Load Program and browse to the COFF output file volume.out. The location of the Probe Points are line 62 (dataInput();), line 70 (dataOutput();), and line 73 (puts("Connecting…..");).
2. Go to File ⇒ File I/O. In the File I/O dialog box, select the File Input tab and select indata.dat. In the Address field, type inp_buffer; in the Length field, type 1; and enable Wrap Around.
3. Click Add Probe Point and select volume.c line 62. In the Probe type field, select Probe at Location. In the Connect To field, select FILE IN: C:\..\indata.dat, click Replace, and click OK.
4. Select the File Output tab and add the outdata.dat file. In the Address field, type out_buffer; in the Length field, type 1.
5. Click Add Probe Point and select volume.c line 70. In the Connect To field, select FILE OUT: C:\..\outdata.dat, click Replace, click OK, and click OK to exit the File I/O dialog box.

6. Go to Debug ⇒ Probe Point and select line 73. In the Probe type field, select Probe at Location if expression is TRUE; in the Expression field, type Input_File(). Click Replace and click OK.

7. You may notice a delay (if the delay is negligible, then no breakpoint is necessary at line 54 and you can run the application freely) between files being copied to the time they are ready in the input buffer. This is due to the fact that we are using the same input file for all of the different test vectors. Since CCStudio holds all input/output data in a queue, once the first file's data has been transferred, the second data is waiting in the queue. For our particular example, if we run the application freely the output has about 4 zeros between each files. To avoid the delay and for contiguous data flow, the developer has the following options:

   a. A breakpoint has to be added after each file transfer so the user can manually forward the data in the input tape play by the appropriate amount of points (in this case 4). An appropriate place to set a breakpoint is at line 54 (puts("volume example started\n");). As a check, one may want to run the application freely to see how many zeros are between the files and then adjust the results, accordingly.

   b. After each file transfer, add a breakpoint to halt the processor, go to File ⇒ File I/O. In the File I/O dialog box, select the File Input tab, remove the indata.dat file, and then reconnect it again with the same previous parameters. Close the File I/O dialog box, continue to run the program, and repeat this for the number of input files.

   Steps 7a and 7b are not illustrated in this application report, since the purpose of this report is to show GEL automation. These two workarounds would require user interface during the running of the application, thus would defeat the purpose of the automation.

8. The delay can vary from one computer to another. On this particular machine, which the application report was based upon, there is a constant delay of 4 zeros. However, depending on the developers' machine, the delay may be higher or lower and it may not necessarily be constant. Note that these delays count as data points since they are being processed from the input to the output buffer. In a real application, the developer may have over 100 data points in each of the input files, in this case the delay may or may not increase in proportion with the file size. To show how GEL can be used to automate testing, the application should require the least amount of user interface. Therefore, a 4^{th} input data file is processed filled with zeros to flush the queue of any data it may contain. The developer is asked to ignore the delay of zeros between each file transfer.

9. Now we are ready to run and test the program. Go to Debug ⇒ Run. Watch your output screens to check the status of the test. 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. Go to GEL ⇒ GEL_Automation ⇒ Test_File1. Recall this GEL function is created using the hotmenu keyword and is part of the volume_test.gel file.

10. Verify the results by checking the contents of the data files. All of the data from the three files is there, except their positions have been pushed down. For example, at the end of the test, the contents of indata4.dat should have been copied into indata.dat. Also, for this particular machine, the resultant output file outdata.dat is given in Appendix F. After rearranging, the developers screen should look similar to Figure 6.

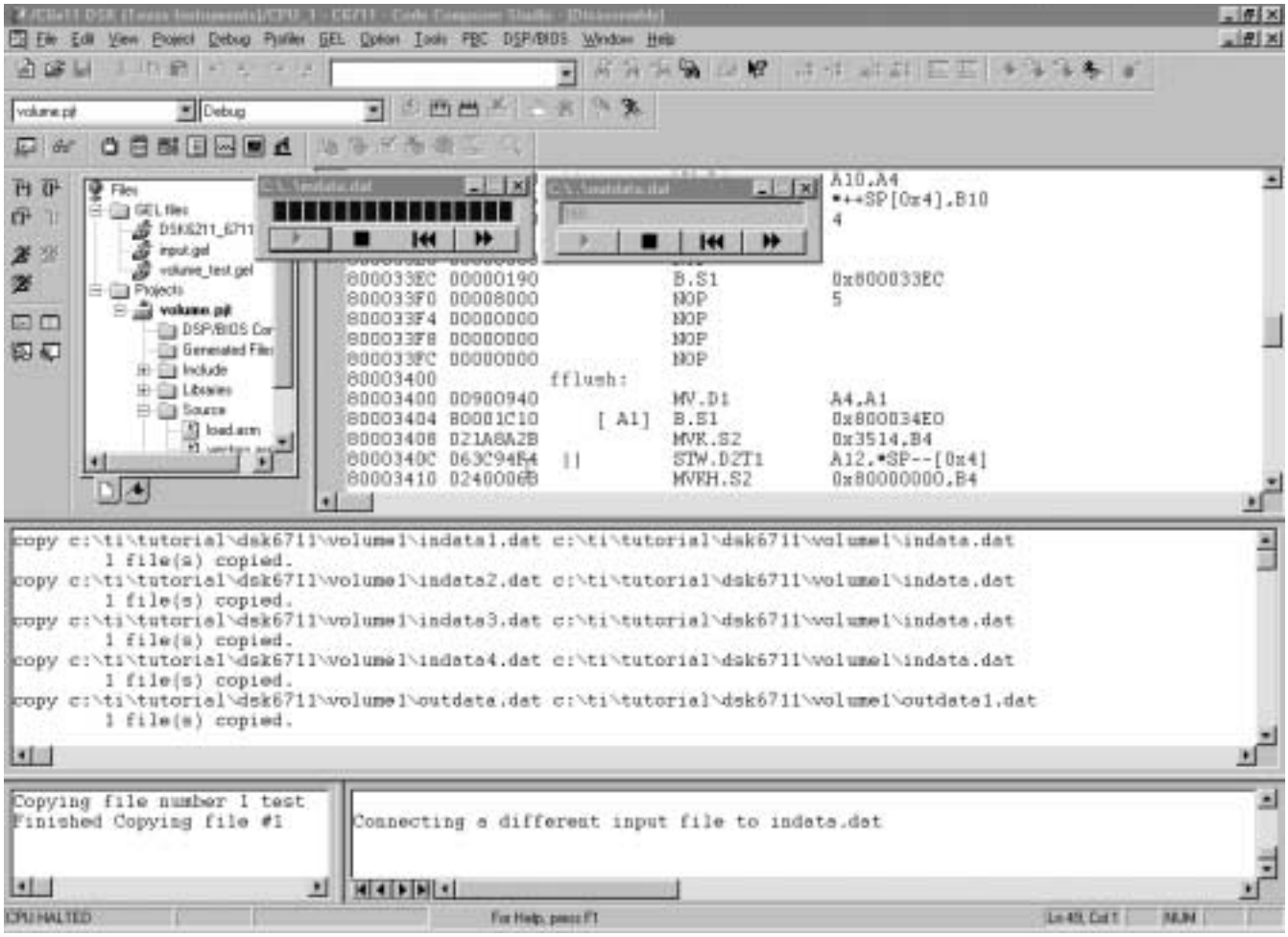

Figure 6. Output Screen of Multiple Vector Case

5 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:

- OnFileLoaded(). If defined in a loaded GEL file, this function is called after a program is loaded into memory. This function requires two integer parameters, one that indicates a success or fail and the other to indicate if only symbols were loaded.
- OnPreFileLoaded(). This function is a counterpart to OnFileLoaded() and if defined in a loaded GEL file, it is called before a program is loaded. This function requires no input parameters.
- OnReset(). This function is called when a target processor has been reset. This function requires one input parameter to suggest if the call to this function was successful.
- OnRestart(). This function is called when the program is restarted. This function requires one input parameter to indicate if the call to this function was successful.

6 Conclusion

A question that commonly arises is why would we 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 startup file. If these files were not loaded, then every time we started CCStudio, the project along with the associated GEL file(s) would manually have to be loaded. This, in turn, not only becomes time consuming but tedious. GEL is particularly important in automating some common tasks and testing the results of a project. As we have shown, given an input file(s), GEL can buffer out these input 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 application report, many built-in GEL functions were used and explained. CCStudio v2.0 comes with over 65 built-in functions ready for the developer to take advantage of. For more information on these, see the online help or the contents menu under help in CCStudio.

This application report hopefully provided the developer with some additional knowledge of GEL and showed what types of features are available to them. Although the application itself is general, the developer should definitely extend the examples provided to fit their specific needs.

Appendix A 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 2.2 and is intended for those
//developers who wish to create their own custom startup file rather than using the
//board specific one

StartUp()

//anything defined in this function will be executed upon load

{

//Load the volume.pjt project

GEL_ProjectLoad(”C:\\ti\\tutorial\\dsk6711\\volume1\\volume.pjt”);

GEL_ProjectRebuildAll();   //Rebuilds all of the files associated with the project

GEL_LoadGel(”C:\\ti\\tutorial\\dsk6711\\volume1\\volume_test.gel”);

}
```

Appendix B volume_test.gel

```
Startup()
{
//The following two code lines are required only for section 2.1 users
//Section 2.2 users should comment out the following two code lines
   GEL_ProjectLoad(”C:\\ti\\tutorial\\dsk6711\\volume1\\volume.pjt”);
   GEL_ProjectRebuildAll();
//Section 4 users should comment out the GEL_System line
GEL_System(“copy c:\\ti\\tutorial\\dsk6711\\volume1\\indata1.dat
c:\\ti\\tutorial\\dsk6711\\volume1\\indata.dat”);
}
menuitem ”GEL_Automation”  //Creates an entry under the GEL drop down menu
hotmenu Run_Test()         //Adds a function under “GEL_Automation”
{
//The developer could define a function that compares the output with the
//expected output
}
hotmenu Test_File1()    //Adds a function under “GEL_Automation”
{
//Displays a message in a default output window
GEL_TextOut(”Copying file number 1 test\n”);
GEL_System(”copy c:\\ti\\tutorial\\dsk6711\\volume1\\outdata.dat
c:\\ti\\tutorial\\dsk6711\\volume1\\outdata1.dat”);
GEL_TextOut(”Finished Copying file #1\n”);
}
```

Appendix C 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++;
   }
//The following code line is only used in section 4.  Comment it out for section 3
   puts(”Connecting a different input file to indata.dat\n”);

//The following code line is only used in section 3. Comment it out for section 4
   puts(”Connecting an Output File to a user defined data file\n”);
}
```

```
/*
 *  ======== 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 D Input Data Files

```
//File 1
//Data for Indata1
1651 1 0 0 0   //Data file header with magic number 1651. For more information on
9              //file headers, 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
```

```
//File 4
//Data for Indata4. This file is used to flush the input queue of any data it may have
1651 1 0 0 0
0
0
0
0
0
0
0
0
0
0
```

Appendix E input.gel

```
//When this GEL file is first loaded into program memory, the StartUp function is
//called and indata1.dat is copied into indata.dat and a 0 is assigned to a free
//memory location on the dsk6711
//Note: 0xFE00 is a free memory location only on the dsk6711, please modify the
//code for your specific target
StartUp()
{
   GEL_System("copy c:\\ti\\tutorial\\dsk6711\\volume1\\indata1.dat
   c:\\ti\\tutorial\\dsk6711\\volume1\\indata.dat");
   *((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 copied
      GEL_System("copy c:\\ti\\tutorial\\dsk6711\\volume1\\indata2.dat
      c:\\ti\\tutorial\\dsk6711\\volume1\\indata.dat");
      GEL_Restart(); //This function restarts the target application
      GEL_Go(main);  //from main
      GEL_Run();     //and run the program loaded on the target
   }

   if(counter==1)    //after executing the first if loop, the counter is
   {                 //incremented and indata3 is copied into indata
      GEL_System("copy c:\\ti\\tutorial\\dsk6711\\volume1\\indata3.dat
      c:\\ti\\tutorial\\dsk6711\\volume1\\indata.dat");
      GEL_Restart();
      GEL_Go(main);
      GEL_Run();
   }

   if(counter==2)    //This loop is added just to flush the input queue. The input
   {                 //file contains all zeros
      GEL_System("copy c:\\ti\\tutorial\\dsk6711\\volume1\\indata4.dat
      c:\\ti\\tutorial\\dsk6711\\volume1\\indata.dat");
      GEL_Restart();
      GEL_Go(main);
      GEL_Run();
   }
                     //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);                      //returns control back to CCSTUDIO so the
                                    //program can further be executed
}
```

Appendix F outdata.dat

```
1651 1 80004290 0 1
0x00000009
0x00000001
0x00000005
0x00000008
0x00000009
0x00000007
0x00000001
0x00000002
0x00000005
0x00000004
0x00000000
0x00000000
0x00000000
0x00000000
0x00000009
0x00000001
0x00000005
0x00000007
0x00000009
0x00000006
0x00000007
0x00000005
0x00000006
0x00000004
0x00000000
0x00000000
0x00000000
0x00000000
0x00000004
0x00000001
0x00000006
0x00000006
0x00000001
0x00000006
0x00000005
0x00000008
0x00000005
0x00000009
0x00000000
0x00000000
0x00000000
```

IMPORTANT NOTICE

Texas Instruments and its subsidiaries (TI) reserve the right to make changes to their products or to discontinue any product or service without notice, and advise customers to obtain the latest version of relevant information to verify, before placing orders, that information being relied on is current and complete. All products are sold subject to the terms and conditions of sale supplied at the time of order acknowledgment, including those pertaining to warranty, patent infringement, and limitation of liability.

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

Customers are responsible for their applications using TI components.

In order to minimize risks associated with the customer's applications, adequate design and operating safeguards must be provided by the customer to minimize inherent or procedural hazards.

TI assumes no liability for applications assistance or customer product design. TI does not warrant or represent that any license, either express or implied, is granted under any patent right, copyright, mask work right, or other intellectual property right of TI covering or relating to any combination, machine, or process in which such products or services might be or are used. TI's publication of information regarding any third party's products or services does not constitute TI's approval, license, warranty or endorsement thereof.

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. Representation or reproduction of this information with alteration voids all warranties provided for an associated TI product or service, is an unfair and deceptive business practice, and TI is not responsible nor liable for any such use.

Resale of TI's 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, is an unfair and deceptive business practice, and TI is not responsible nor liable for any such use.

Also see: Standard Terms and Conditions of Sale for Semiconductor Products. www.ti.com/sc/docs/stdterms.htm

Mailing Address:

Texas Instruments
Post Office Box 655303
Dallas, Texas 75265