Application Report
SPRAA74A – December 2004

Creating Device Initialization GEL Files

Alan Campbell, Ki-Soo Lee, Dan Rinkes, Darian Sale *SDS Applications Engineering*

ABSTRACT

Startup GEL files are used to automate device initialization when Code Composer Studio starts up. As devices become more complex, the startup GEL files also increase in complexity, making it more difficult to write proper startup GEL files. This application note describes how Code Composer Studio uses the startup GEL files and provides guidelines on how to properly write such files.

This application note focuses on CCStudio versions 2.40 and greater.

Contents

Tables

1 Introduction

The General Extension Language (GEL) can be used to configure the Code Composer Studio development environment and to initialize the target CPU. GEL is an interpreted language, and its syntax is similar to that of C. A rich set of built-in GEL functions is available, or you can create your own GEL functions.

This document describes ways to create well-written GEL startup files. For a summary of recommendations, see *Appendix B: GEL Guidelines.*

2 Code Composer Studio Startup Process

In the CCStudio setup utility, you can associate a GEL startup file with each processor in your system configuration. When CCStudio is launched, startup GEL files defined in the CCStudio setup utility are loaded into the host PC's memory and the "StartUp()" function is executed (if one is defined).

In older versions of CCStudio (v2.3 or earlier), both host and target initialization steps were often performed in the StartUp() function to automate the initialization process. However in versions of CCStudio that support Connect/Disconnect (see *Appendix A: CCStudio Version Support Matrix*), such GEL files may not work properly since CCStudio starts up disconnected from the target. This causes actions to fail in StartUp() if they attempt to access the target. A new built-in GEL callback function called OnTargetConnect() has been provided to perform target initialization to put the target in a good state. CCStudio calls this function after a connection with the target has been established. Any initialization steps that do not access the target may remain in StartUp().

NOTE: If CCStudio v2.4 and v3.1+ are configured to use a simulator, OnTargetConnect() is called right after StartUp() since CCStudio is always "connected" to the simulator target. OnTargetConnect() is not automatically called in CCStudio v3.0 when using a simulator. See *Appendix B: GEL Guidelines* for more details.

3 Using the GEL Callback functions

Proper use of the built-in GEL callback functions is essential for writing proper startup GEL files. While StartUp() and OnTargetConnect() are the most commonly defined callback functions, it is important to be aware of the functionality of all the callback functions described in the following subsections.

3.1 StartUp() Function

The StartUp() GEL function is called by CCStudio whenever the GEL file containing it is loaded. Since startup GEL files are loaded upon CCStudio startup, the StartUp() function is run at that time. Versions of CCStudio before 2.40 performed both host and target initialization here. If you are using a version of CCStudio that supports Connect/Disconnect, no actions that attempt to access the target can be placed here since CCStudio starts disconnected from the target. Place target initialization actions in the OnTargetConnect() callback described in Section 3.2.

Behavior recommended and not recommended for the StartUp() function is as follows:

Recommended:

- Setting up basic CCStudio memory map (that does not need target access to accomplish). See Section 4 for more information on memory maps.
- Any basic initialization step that does not attempt to access the target.

Not Recommended:

- GEL_Reset() (This attempts to access the target, doing a device reset via emulation.)
- Setting Breakpoints via GEL_BreakPtAdd(). (This also tries to access the target.)
- GEL_TextOut() and GEL_OpenWindow(). Since StartUp() executes before any CCStudio control window is open, this may cause an error depending on the version of CCStudio.
- Basically *any* action that attempts to access the target.

Here is an example of the StartUp() call in the DM642EVM.gel file *before* Connect/Disconnect was supported:

```
/*-----------------------------------------------------------*/
/* The StartUp() function is called each time CCS is started.   */
/* Customize this function to perform desired initialization.   */
/*-----------------------------------------------------------*/

StartUp()
{
    setup_memory_map();
    GEL_Reset();  /* Do not call in StartUp() with CCStudio v2.4 or higher */
    init_emif();  /* Do not call in StartUp() with CCStudio v2.4 or higher */
}
```

Here is an example the StartUp() call in the DM642EVM.gel file modified to support versions of CCStudio that support Connect/Disconnect:

```
/*-----------------------------------------------------------*/
/* The StartUp() function is called each time CCS is started.   */
/* Customize this function to perform desired initialization    */
/* that will not access the target.                             */
/*-----------------------------------------------------------*/

StartUp()
{
    setup_memory_map();
}
```

Note that both GEL_Reset() and the custom function init_emif() (which initializes the EMIF) must be removed because both calls try to access the target. Remember that any actions that attempt to access the target are not allowed in StartUp(). The first thought would be to move those calls to OnTargetConnect(). But do they really belong there? The next section answers that question.

The custom function setup_memory_map() can remain since that call simply sets up the CCStudio memory map and does not try to access the target in any way.

3.2 OnTargetConnect() Function

With the introduction of Connect/Disconnect in CCStudio version 2.40, the OnTargetConnect() callback was created with the intention that it perform the minimum target initialization needed to access the target regularly (for example, to set PLLs and disable watchdog timers). Its execution finishes before anything else occurs.

You might tend to view OnTargetConnect() as a location for *all* desired target initialization steps that cannot be run in StartUp(). We would like to discourage this for the following reasons:

- **If using CCS 2.40 and 3.0:** Except for GEL_Reset() and GEL_IsInRealtimeMode(), *calls to any built-in GEL functions that access the target are not allowed.* Calls to custom GEL functions that do not call built-in GEL functions themselves are fine. These built-in GEL calls are not recommended because some built-in calls can disrupt the Connect/Disconnect process in CCStudio. If GEL built-in calls that access the target are needed for additional initialization, it is recommended that they be called from a separate function after a target connection has been established and after OnTargetConnect() has finished execution. For example, such a function could be called from the GEL pull-down hotmenu.

  NOTE: For CCStudio 3.1 (and higher), this limitation does not exist because the OnTargetConnect() callback has been modified to be called later when it will not disrupt the connect procedure. Any target initialization action can be made here if using CCStudio 3.1

- It is important to remember that OnTargetConnect() is called each time a target connection is established. If you disconnect the target and do a power cycle, re-initialize the target upon re-connecting. Treat every connection as a first connection—with the essential target initialization actions always run. Otherwise, people may disconnect, power cycle the target, try to connect, and run into problems. If any custom target initialization steps should *not* be run every time a target connection is made, these steps should be placed in a custom GEL function that can be called manually as described in the preceding paragraph.

If you connect to the target in real-time mode, you can connect unobtrusively and leave the target in a running state. Issuing a command in OnTargetConnect() that modifies the target state (such as GEL_Reset()), halts the target and defeats the ability to connect unobtrusively. Real-Time mode is available on 'C27x, 'C28x, 'C55x and 'C64x processors.

There is a new built-in GEL function called GEL_IsInRealtimeMode(). (See *Appendix A: CCStudio Version Support Matrix.*) This call returns a 1 if the target is started up in real-time mode, and 0 if it not. This call can be used in OnTargetConnect() to perform different actions depending on whether you are in real-time mode.

```
/* OnTargetConnect() is called each time a target is connected.        */
/* Its execution finishes before anything else occurs.                 */
/* Customize this function to perform essential target initialization. */
OnTargetConnect()
{
        // Check to see if started up in real-time mode
        if (GEL_IsInRealtimeMode() ) {
               // do real-time target init stuff
        } else {
               // do regular initialization
               GEL_Halt();
               GEL_Reset();
        }
}
```

NOTE: Even if you configure CCStudio to connect to the target automatically when you open a control window, StartUp() is still called before the target is connected. OnTargetConnect() should always be used to automate critical target initialization steps.

Behavior recommended and not recommended for the OnTargetConnect() function is as follows:

Recommended:

- Absolute minimum target initialization actions to get the target in a reliable state for CCStudio. For example, disabling watchdog timers and pulling the DSP out of reset in heterogeneous environments.

Not Recommended:

- **If using CCS 2.40 and 3.0:** Calls to *any* built-in GEL function that *accesses the target* (for example, GEL_MemoryFill(), GEL_BreakPtAdd(), and GEL_Load()) with the exception of GEL_Reset() and GEL_IsInRealtimeMode(). As mentioned earlier, CCStudio 3.1 does not have this limitation.
- Non-critical actions that should not be repeatedly called each time there is a target disconnect and connect.
- "Custom" non-essential target initialization steps.

In summary, the default OnTargetConnect() function should only call what is necessary. Give users a baseline initialization.

The OnTargetConnect() call from the DM642EVM.gel is shown below. Note that both the GEL_Reset() call and the init_emif() calls removed from StartUp() are performed here. This is acceptable for CCStudio version 3.1 and greater.

```
/*------------------------------------------------------------*/
/* OnTargetConnect() is called every time a target is connected.*/
/* Its execution finishes before anything else occurs. Customize*/
/* this function to perform essential target initialization.    */
/*------------------------------------------------------------*/
OnTargetConnect()
{
        // place critical target initialization steps here
        GEL_Reset();
        init_emif();
}
```

For some platforms, calling GEL_Reset() may be necessary to put CCStudio in a "good" state for the target. You should test to see whether this call is necessary. It is recommended that you minimize the complexity of GEL startup functions—including the number of calls to GEL_Reset()—where possible.

3.3 OnPreFileLoaded() Function

This callback function is called before a program/symbol (.out) file is loaded. This is a good place to perform additional target initialization that needs to be done before the program can be loaded and debugging done. For example, you can initialize external memory here.

```
/*-------------------------------------------------------------*/
/* OnPreFileLoaded()                                           */
/* This function is called automatically when the 'Load Program'*/
/* Menu item is selected.                                      */
/*-------------------------------------------------------------*/
OnPreFileLoaded()
{
       FlushCache();
       IER = 0;
       IFR = 0;
       init_emif();
}
```

3.4 OnFileLoaded() Function

This callback function is called after a program/symbol file has been loaded. Actions such as setting up the debug source search path (if no CCStudio project file exists), setting breakpoints and probepoints, performing a software reset and restart, and loading a saved profile configuration can be done here. Note that the software reset and restart is a necessary common step on many targets. See *Appendix B: GEL Guidelines* for an example that uses OnFileLoaded().

3.5 OnReset() Function

This callback function is called after the target processor has been reset. If you need to restart the program each time you do a software reset, the GEL_Restart() call can be made here. An example from the DM642EVM.gel file that uses OnReset() is shown below:

```
/*-------------------------------------------------------------------------*/
/* OnReset()                                                               */
/* This function is called automatically after a SW Reset has been executed. */
/*-------------------------------------------------------------------------*/
OnReset(int nErrorCode)
{
    init_emif();
}
```

3.6 OnRestart() Function

This callback function is called after the program is restarted. The following is an example of OnRestart() from the DM642EVM.gel file:

```
/*-------------------------------------------------------------------------------*/
/* OnRestart()                                                                   */
/* This function is called by CCS when you do Debug->Restart. The goal is to put the */
/* C6x into a known good state with respect to cache, edma and interrupts. Failure   */
/* to do this can cause problems when you restart and run code multiple times.       */
/*-------------------------------------------------------------------------------*/
OnRestart(int nErrorCode )
{
      /* Turn off L2 for all EMIFA CE spaces.  App should manage these for coherency */
      GEL_TextOut("Turn off cache segment\n");

      *(int *)0x1848200 = 0;  /* MAR0 */
      *(int *)0x1848204 = 0;  /* MAR1 */
      *(int *)0x1848208 = 0;  /* MAR2 */
      *(int *)0x184820c = 0;  /* MAR3 */

      /* Disable EDMA events and interrupts and clear any pending events. */
      GEL_TextOut("Disable EDMA event\n"); */

      *(int *)0x01A0FFA8 = 0;          /* CIERH */
      *(int *)0x01A0FFB4 = 0;          /* EERH */
      *(int *)0x01A0FFB8 = 0XFFFFFFFF; /* ECRH */

      *(int *)0x01A0FFE8 = 0;          /* CIERL */
      *(int *)0x01A0FFF4 = 0;          /* EERL */
      *(int *)0x01A0FFF8 = 0xFFFFFFFF; /* ECRL */

      /* Disable other interrupts */
      IER = 0;
      IFR = 0;
}
```

3.7 OnHalt() Function

This callback function is called each time the CPU is halted. You can use the OnHalt() callback for actions such as logging the values of variables and registers to GEL_TextOut().

4 Memory Mapping

The CCStudio memory map tells the debugger which areas of target memory it can and cannot access. If accessing invalid memory causes problems on the hardware, memory mapping allows you to tell the driver not to access it—making memory mapping an effective way to avoid emulation errors. Setting up the CCStudio memory map is often done in the StartUp() callback function of the device startup GEL file using the following built-in GEL functions.

4.1 GEL_MapAdd() Function

The GEL_MapAdd() function is the most common function used in startup GEL files. This function adds a section of memory to the memory map. You can specify the starting address, length, memory page (program/data/IO), and readable/writeable flags. Correct memory maps can help avoid frustration during debugging. CCStudio knows what memory it can access and what memory it cannot access exclusively through the memory map.

One of the most common problems that can be solved using memory mapping is a section that is inadvertently linked to a location where no memory exists. If the memory map shows the hole in memory, CCStudio can warn a user if an attempt is made to load code into this section.

If all memory is of the same type and access size, GEL_MapAdd() should be sufficient for defining a memory map.

4.2 GEL_MapAddStr() Function

GEL_MapAddStr() is a superset of the GEL_MapAdd() function. It provides the same readable/writable attributes that can be specified in GEL_MapAdd(). However, it also supports additional parameters for memory access size, a “shared memory” tag, and specification of memory wait states. With the addition of GEL_MapAddStr(), there is really no reason to use GEL_MapAdd(). GEL_MapAdd() is maintained for compatibility purposes.

For example, one occasion where GEL_MapAddStr() is useful is when 32-bit accessible external memory is being used, such as a 32-bit accessible SDRAM. If GEL_MapAddStr() is not used, when CCStudio accesses this memory to fill memory and disassembly windows, it uses the default access size is. By specifying in the GEL file that this memory is 32-bit addressable only, the emulation driver will make the proper sized access.

```
GEL_MapAddStr(0x80000000, 0, 0x02000000, "R|W|AS4", 0);   // 32 bits SDRAM
```

4.3 GEL_MapDelete() Function

GEL_MapDelete() allows a portion of the memory map to be removed. This can be useful if it is necessary to have a “dynamic” memory map. If at certain points some portion of memory is not available, you can remove these map sections so that the debugger does not try to access them.

GEL_MapDelete() accepts only two parameters: address and page. If there are two (or more) adjacent memory sections on the same page with the same parameters, performing GEL_MapDelete() on an address in either (any) of the sections will delete all of these sections. CCStudio tracks adjacent memory sections created with the same parameters to GEL_MapAdd() as a single memory section.

4.4 GEL_MapOn() and GEL_MapOff() Functions

If at some point you need to turn on or off the memory mapping feature, you can do so with the GEL_MapOn() or GEL_MapOff() functions. When mapping is turned off, CCStudio does not respect any of the settings in the memory map. It assumes that it has access to the entire memory space.

4.5 GEL_MapReset() Function

GEL_MapReset() can be used to clear all of the settings in the memory map configuration. With no memory map, the default setting is that none of the memory space is accessible.

5 Avoiding GEL Initialization Entirely for Production Applications

Relying on a GEL file to perform tasks such as EMIF initialization can become a crutch. When your application nears production, the settings made in GEL—for example, to configure the SDRAM timing and to refresh—need to be moved to your bootload code. For this reason, you may want to consider creating a GEL file that sets up the memory map via GEL_MapAdd() to enable CCStudio debugging, but that no longer performs peripheral settings such as EMIF writes or watchdog disabling.

Since GEL syntax matches standard C syntax, much of a typical init_emif() GEL routine can be moved to a .c file and linked with the application. However this approach has some caveats:

- Moving GEL syntax to C doesn't work for initialized code (for example, in the .text section) and data (for example, in the .const section). Thankfully the debugger lets you know this. At load time CCStudio performs a data verification of any initialized code/data (that is, anything that needs to be loaded) that was written to memory. The data verification fails because the EMIF has not yet been set up.
- Moving GEL syntax to C works fine for un-initialized data such as heaps. However, you need to ensure the init_emif() C function is called before the heap is used. In a DSP/BIOS application, you can ensure this by plugging the GBL user init function since this gets called early in the code startup sequence.
- It's a good idea to add "volatile" qualifiers as follows to ensure that the Codegen Optimizer doesn't optimize the memory references out.

```
*(volatile int *)EMIFA_SDRAMTIM = 0x00000618;  /* SDRAM timing (refresh) */
```

Finally, these tips are useful if you want to stay in a CCStudio build/debug environment but avoid GEL file peripheral settings. When you do reach the production timeframe, you'll need a smart loader booting from flash or a host machine that sets up EMIF and then copies runtime-critical initialized code/data sections via DMA or memcpy(). Several application notes describe how to achieve this with new Codegen utilities, Hex converters etc. [Reference 2]

6 References

1. *Code Composer Studio v3 Online Help (SPRH199 bundled with the software)*
2. *Creating a Second-Level Bootloader for FLASH Bootloading on C6000* (SPRA999)

Appendix A: CCStudio Version Support Matrix

The chart below describes which versions of CCStudio supports various new GEL capabilities:

Table 1. CCStudio Version Support Matrix

CCStudio Version	OnTargetConnect() (and Connect/Disconnect)	GEL_IsInRealtimeMode()	GEL Global Variables	GEL Relative Path Macro
Pre-2.40 [1]	No	No	No	No
2.40 [2]	Yes [3]	No	No	No
3.00 [4]	Yes	No	No	No
3.10 [5]	Yes [6]	Yes	Yes	Yes

If you are using a device or version of CCStudio that does not support OnTargetConnect() and Connect/Disconnect, then essential target initialization actions can be called in StartUp() since CCStudio always attempts to connect to the target before running StartUp().

GEL global variables and the relative path macro are briefly described in *Appendix B: GEL Guidelines*.

[1] Applies to all ISAs.
[2] 2.40 is an OMAP release for ARM7/9, 'C54x, and 'C55x only.
[3] 'C54x not supported with TI emulators. Supported by Spectrum Digital Emulators.
[4] 3.00 is a C6000 release only
[5] Applies to all ISAs
[6] Connect/Disconnect is not supported by TI emulators on 'C54x and 'C24x. Supported by Spectrum Digital Emulators.

Appendix B: GEL Guidelines

It is recommended that you use the following guidelines when creating GEL files:

- **Add well-written "C" style comments.** As startup GEL files become more complex, well-written comments describing exactly what is happening become even more important. Without well-written comments, it is difficult for people to *properly* modify GEL files without running into future errors. This includes both function-level comments and source-level comments
- **Ensure the GEL file is in "DOS" format.** This means DOS-style carriage returns, etc. Utilities such as unix2dos can be used to get rid of pesky ^M characters.
- **Define all built-in GEL callback functions.** Define functions even if the body of the function is empty. This helps to clarify that when the callback occurs, nothing happens. See Section 3 for all the GEL callback functions available.
- **Notify users of all target initialization action done by CCStudio.** Use GEL_TextOut() to provide feedback for all target initialization done by CCStudio. This helps users keep track of all target initialization actions.
- **Avoid using absolute paths in your GEL files to allow for maximum portability (CCStudio v3.1+).** Relative paths are supported in versions of CCStudio 3.1 and higher with the new GEL macro '$(GEL_file_dir)'. This macro contains the path to the current GEL file. An example of proper macro usage is found below:

```
DebugSourceSearchPaths()
{
       // Source search paths
       GEL_SrcDirAdd( "$(GEL_file_dir)\\..\\..\\sourcedir1" );
       GEL_SrcDirAdd( "$(GEL_file_dir)\\..\\..\\sourcedir2" );
}
```

- **Don't name a GEL global variable something a user might place in their code (CCStudio v3.1+).** Target symbol names take precedence over GEL global variables names. It is recommend that you name GEL global variables using a format of gGEL_*variableName* to avoid the risk of users placing the same variable name in their code.
- **Don't put your "pet GEL stuff" in mainline board GEL file.** Keep your <*board*>.gel file clean and hence reusable. Don't put user-specific preferences in the board GEL file. For example, if you need to enable source-level debug of C files in many different directories, it is best to put your calls to GEL_SrcDirAdd() in a myAppDebug.gel file and then simply perform GEL_LoadGel() after loading the board GEL file. Note that you cannot have multiple definitions of the same GEL function. So if you are loading a separate GEL file (myAppDebug.gel), make sure that this GEL file does not redefine GEL functions in the board GEL file (<*board*>.gel). If you want to automate custom actions using GEL callbacks, add a single call to a custom GEL function in the callback found in the board GEL file. This minimizes the impact to the main board GEL file.

For example, suppose the <*board*>.gel file defines the GEL callback function OnFileLoaded() as follows:

```
/*--------------------------------------------------------------------*/
/* OnFileLoaded() is called after a program is loaded.                */
/* Customize this function to automate actions before a               */
/* program is loaded                                                  */
/*--------------------------------------------------------------------*/
OnFileLoaded (int nErrorCode, int bSymbolsOnly)
{
   // check for errors in loading program
   if (nErrorCode)  {
      GEL_TextOut("An error occurred while loading a file. -%d-\n",,,,, nErrorCode);
   } else    {
      GEL_TextOut("File was loaded successfully. -%d-\n",,,,, nErrorCode);
   }

   // Check to see if only symbols are loaded
   if (bSymbolsOnly) {
      GEL_TextOut("Only symbols were loaded.\n");
   } else{
      GEL_TextOut("Full load.\n");
}

   // Add my source search paths
   DebugSourceSearchPaths();     // this function is defined in myAppDebug.gel
}
```

The myAppDebug.gel file then contains the custom action to add the debug source search paths as follows.

```
hotmenu DebugSourceSearchPaths()
{
        // Source search paths
       GEL_SrcDirAdd( "$(GEL_file_dir)\\..\\..\\sourcedir1" );
       GEL_SrcDirAdd( "$(GEL_file_dir)\\..\\..\\sourcedir2" );
}
```

The myAppDebug.gel GEL file can be auto-loaded by calling GEL_LoadGel() in the StartUp() callback function of the main <*board*>.gel file as follows:

```
GEL_LoadGel("$(GEL_file_dir)\\..\\..\\myprojects\\customGEL\\myAppDebug.gel")
```

- **Place only the minimum necessary initialization actions in StartUp() and OnTargetConnect() (CCStudio v2.40+).** As discussed earlier, place just the minimum actions to put both the host and target in a "good" state in StartUp() and OnTargetConnect(). Also, since users can disconnect and re-connect to the target several times during a debug session, do not put any actions on OnTargetConnect() that should not automatically be executed each time a connection is established to the target.

- **Create GEL hotmenu items for all automated target initialization actions.** Even though some target initialization actions are done in GEL callback functions, it is recommended to create GEL hotmenu items to allow users to manually call these actions. Users should always have the option to call all initialization actions manually.

- **Use GEL_MapAddStr() instead of GEL_MapAdd().** GEL_MapAddStr() provides the ability to define more attributes for the memory region being added—for example, the access size and wait states. The more information provided to CCStudio about the attributes of the memory regions, the less chance of error.

- **Treat all GEL startup files as if connecting to an actual target (CCStudio v2.40+).** When using a simulator, CCStudio is always be "connected" to the simulated target. Thus adding actions that touch this simulated target in StartUp() would work fine. However we still *strongly* encourage people using a simulator to define such actions in the OnTargetConnect() callback function as if they were using actual hardware. This helps startup GEL files be consistent in behavior and makes GEL files more portable between simulator and hardware. When using CCStudio v3.0, which does not automatically call OnTargetConnect() when using a simulator, a good idea is to explicitly call OnTargetConnect() in the StartUp() function. For example:

```
/*------------------------------------------------------------*/
/* The StartUp() function is called each time CCS is started.   */
/* Customize this function to perform desired initialization    */
/* that will not access the target.                             */
/*------------------------------------------------------------*/
StartUp()
{
    /* setup CCS memory map */
    setup_memory_map();

    /* This explicit call is needed for CCS v3.0 when using a simulator */
    OnTargetConnect();      /* comment out this call when using v2.40 and v3.1+ */
}
```

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