Application Report
SPRAC58–November 2016

# KeyStone™ I Bootloader Resources and FAQ

**ABSTRACT**

The KeyStone I Bootloader Resources and frequently asked questions (FAQ) document aims to combine the information about the C66x DSP bootloader that is available through data manuals, user guides, and SDK documentation for C667x and C665x. It also provides examples, tips, and tricks to boot application images on the TI supported EVMs and provides additional pointers on modifying the booting for custom boot scenarios. This document also contains a collection of responses to frequently asked questions on the TI E2E™ online community forums regarding the DSP bootloader.

**Contents**

## Trademarks

E2E, KeyStone, Code Composer Studio are trademarks of Texas Instruments.

## 1 Bootloader Documentation

The documentation for the KeyStone™ I bootloader is distributed across three resources.

Common user guide for all KeyStone I devices that provides the details of the BOOTROM design and expected behavior of the BOOTROM on each of the supported boot modes in the KeyStone I family of devices. The device covers-in some of the details and the behavior of the device after different reset and hibernation modes. It will also provide a preview of how to create boot images.

- *KeyStone Architecture DSP Bootloader*

The following data sheets provide boot mode settings, parameter tables, DDR configuration tables, and memory maps specific to the devices.

- TMS320C6678 (SPRS691)
- TMS320C6670 (SPRS689)
- TMS320C6655 and TMS320C6657 (SPRS814)

The *Keystone ROM Boot Examples and Reference code* section of the *Keystone Device Architecture* wiki provides access to boot ROM source code for reference to understand the implementation.

*Processor SDK RTOS BOOT C66x* — Boot software, examples validated on the TI evaluation module (EVM), the flash writer, and utilities are packaged with the device software development kit (SDK).

## 2 Direct Boot Examples (Without IBL)

The TI C66xx EVM always implements an intermediate boot loader after power on reset. The FPGA firmware on the EVM redirects the core to the intermediate boot loader (IBL) that is flashed on the EEPROM and then directs it to the desired boot mode. In custom designs, adding an additional EEPROM for implementing the IBL may not be practical because of cost and size reasons. The PG 2.0 silicon for all the C66x devices have the PLL lockup issue fixed and no longer require implementing an IBL on custom boards.

To demonstrate that the hardware can be booted directly from the ROM boot loader without the IBL, TI has created some examples that can be used to validate this flow on the EVM hardware. The two examples described below are for SPI boot on C6678 and NAND boot on C6657. Each of the examples have a ReadMe.txt file or other document that walks users through the processing of creating these boot images.

- For documentation of these examples, refer to the user manual and ReadMe.txt files provided in the docs folder of the three packages.

# 3 Bootloader Utilities

## 3.1 Where can I find the boot utilities?

The boot utilities for KeyStone I devices are packaged as part of the bootloader software package in the Processor SDK RTOS in the following folder path: pdk_<deviceName>_x_x_x\packages\ti\boot.

**NOTE:** In earlier MCSDK for KeyStone I devices, the boot utilities were provided in the directory path mcsdk_2_xx_xx_xx\tools\boot_loader

## 3.2 What bootloader utilities are available for the C66xx devices?

The following links contain information about the bootloader utilities available for C66xx devices.

- *Processor SDK RTOS BOOT C66x*
  - *Bootloader Execution Sequence* section
  - *Flash Writers* section
- *Multicore Application Deployment (MAD) Utilities User's Guide*

## 3.3 What utilities should I use if I am not using a secondary bootloader or IBL?

The boot utilities provided in the utils folder of the IBL package and the bin folder of the C6000 compiler can also be used for creating a boot image if your boot design does not leverage the IBL. Documentation for these utilities is provided as part of the compiler or as part of the SDK offering based on where the utility is found. For typical usage of these utilities, refer to SDK examples or the boot example in Section 3.2.

- **Hex6x utility in C6000 compiler** — converts an object file (.out) into the boot table format required by the DSP boot loader. This utility is documented in Chapter 11 of the *TMS320C6000 Assembly Language Tools v7.4* user's guide included as part of the C6000 compiler.
- **catccs** — concatenates Code Composer Studio™ (CCS) format files.

```
Usage: catccs <infile1> <infile2> [<infile3> [<infile4> [..]]] [-out <outfile>] [-addr <address>]
  <infile1>, <infile2>, <infile3>, <infile4> and <outfile> are .ccs files.
  address - load address for the concatenated ccs file.
```

- **ccs2bin** — converts CCS format to binary

```
Usage:  ccs2bin [-swap] <ccsfile> <binfile>
  <ccsfile> - input .ccs files.
  <binfile> - output .bin files.
```

- **b2ccs** — converts a hex b file into a CCS data file.

```
Usage:   b2ccs [-noorg] <hexfile> <ccsfile>
  <hexfile> - Hexadecimal blob file.
  <ccsfile> - Output .ccs file.
  if -noorg is used there is only one header line
```

- **Romparse** — used to append a boot parameter table to the boot table.

```
Usage:   romparse [-noorg] <hexfile> <ccsfile>
  <hexfile> - Hexadecimal blob file.
  <ccsfile> - Output .ccs file.
```

**NOTE:** The romparse utility in the SDK is only a reference implementation that addresses use of the SDK. The utility may need to be modified if there are some boot parameter table fields missing or if less than eight boot parameter tables need to be attached to the boot table data. The utility is provided in the source to allow for customization.

**NOTE:** The utility hard code the $I^2C$ addresses of EEPROM used on the EVM in the boot parameter table. If users are using SPI boot the address must be set to 00.
The utility is designed to process boot images less than 4Kb in size. Please modify the size parameter in the utility if the boot image is greater than 4 Kb in size.

- **b2i2c**

```
Usage:   b2ccs [-noorg] <hexfile> <ccsfile>
  <hexfile> - Hexadecimal blob file.
  <ccsfile> - Output .ccs file.
  if -noorg is used there is only one header line
```

- **bconvert64x** — used to format the boot table data that is not a multiple of 4 bytes to the correct endian format expected by the boot loader.

```
Usage:   bconvert -be|-le [input_file] [output_file]
  <input_file> - Input Boot table data file
  <output_file> - Output Boot table data file.
```

### *3.4 What is the syntax that needs to be used in an .rmd file that is used to create the boot image?*

The .rmd file is intended to be used with the hex6x tool provided with the C6000 compiler. The syntax to create a binary image using hex6x and the .rmd file has been described in the Hex Conversion Utility Description section of the C6000 compiler documentation.

## 4 How do you load same image across multiple cores? Examples?

View the multi-core boot example given in the SDK under the ~\boot_loader\examples\srio\srioboot_helloworld\src directory. This example shows how users can wake up secondary cores using IPC interrupts and populate the magic address for the cores to start executing code. This structure is useful for creating a single application that boots on all cores; it also has control code to partition itself across secondary cores by checking the core number on which it is running.

The boot ROM will initialize the SOC and load a single image into memory. All cores will run the same image but the code will have control code to check which core is running the image and partition the code. Core0 boots first, so it must populate the entry point of the image in the magic address for all of the secondary cores and issue them an IPC interrupt to wake them up.

## 5 Which gel files can be used to connect C66xx DSP cores and where can I find them?

GEL stands for General Extension Language (previously GODSP Extension Language). GEL is the expression language that is used by the CCS debugger.

Target configurations in CCS often specify a startup script. These scripts are typically used to setup the memory map for the debugger and set any initial target state (through memory or register writes) that is necessary to connect the debugger. These scripts are usually written in GEL. For example, it is the OnStartup() function in the GEL file that is run when the debugger is launched, and an OnTargetConnect() function is called when the target is connected. The startup(gel) scripts define these functions.

These gel files are packaged as part of CCS, and older MCSDK has it packaged in the tools directory.

### *5.1 GEL Files*

In MCSDK 2.x, find the appropriate gel files for KeyStone-I EVMs under the ~\ti\mcsdk_2_0x_0x_0x\tools\program_evm\gel path. In CCSvX, find the gel files under the ~\ti\ccsvX\ccs_base\emulation\boards\evmc66xxl path.

### *5.2 Target Configuration Files*

A default is available here, and users can create as many as required.

In MCSDK 2.x, find the *.ccxml files under the ~\ti\mcsdk_2_0x_0x_0x\tools\program_evm\configs path. In Processor SDK, find the *.ccxml files under the ~\ti\processor_sdk_rtos_c66xx_x_xx_xx_xx\bin\configs\evm66xxl path.

## 6 How do I set the boot modes?

Please refer the *Boot Mode Dip Switch Settings* section of the respective EVM Hardware Setup wiki pages to set the desired boot modes.

- C6657 EVM
- C6670 EVM
- C6678 EVM

## 7 How do you wake up secondary cores from the primary core (core0)?

During the boot process, the boot loader executes an IDLE command on the secondary CorePacs and keeps the secondary CorePacs waiting for an interrupt. After loading the secondary CorePacs application, the BOOT_MAGIC_ADDRESS in individual corePacs are populated, the application code in the corePac0 can trigger the IPC interrupt to wake up the secondary cores and branch up to the address specified in the BOOT_MAGIC_ADDRESS. Please try the following procedure in the EVM.

Source: <a _fcknotitle="true" href="Helloworld.zip">Helloworld.zip</a>

1. Set the EVM to No Boot mode, connect to core0 and use gel file to initialize the core0 (users can run Global_Default_Setup to initialize the PLL and DDR).
2. Go to the memory browser, read the address (0x1187FFFC), and write it down.
3. Build the attached source and load and run on core0.

   After completion of execution, core0 will wait in busy loop(while(1)).
4. Pause the debugging.
5. Go to memory browser, then enter the address 0x1187FFFC → 0xBABEFACE.

This means that core0 wakes up core1 successfully. The value 0xBABEFACE has been written by core1 from function "write_boot_magic_number()". Users can follow this for all secondary cores by updating NUMBER_OF_CORES.

**NOTE:** Make sure to build the application with the appropriate macros for the EVM and magic address in steps 2 and 5. The previous steps are for the C6678.

```
/* Magic address RBL is polling */
#ifdef _EVMC6657L_
#define MAGIC_ADDR 0x8ffffc
#endif
#ifdef _EVMC6678L_
#define MAGIC_ADDR 0x87fffc
#endif
#ifdef _EVMC6670L_
#define MAGIC_ADDR 0x8ffffc
#endif
```

## 8 How do you boot from NAND flash on the C6670 and C6678 EVMs?

The NAND boot on the C6670 and C6678 EVMs requires an intermediate boot loader (IBL), but the custom boards with revision 2.0 system-on-chip (SoC) devices do not require an IBL. The primary motivation for the IBL is because of a PLL workaround that is not present in the basic ROM boot. Users can find references on this in the ibl/src/device directory.

Please refer to the MCSDK user's guide for IBL flashing and NAND boot setup.

## 9 How to boot from NAND flash on the C6657 EVM?

Direct and IBL based NAND booting is possible on the C6657. For IBL booting, please refer to *Processor SDK RTOS BOOT C66x* for NAND boot setup. For direct NAND booting, please find the direct NAND boot example that is validated on C6657 EVM (in the *Direct Boot Examples* section).

## 10 Why is the updated FPGA bit file required for direct NAND booting on the C6657? Where can I get the FPGA bit file?

There is an advisory note in the errata associated with this boot mode that **TI** would like to make users aware of. The following is a summary of the issue:

> Booting from a NAND flash device will fail when the C6657/C6655 is at full speed. The root cause of the problem is that insufficient time is allowed for the NAND device to complete the initial reset command sent to it by the DSP ROM code. After sending the reset command, the DSP executes a wait loop (for time out). This wait loop allows a maximum number of iterations before halting and declaring that the NAND is not responding correctly. The wait loop does not allow enough time for the NAND to complete its initial reset sequence.

### *10.1 Proposed Workaround*

Use the FPGA firmware on the system to apply reset to the NAND flash device, then apply reset to the C6657 device after the NAND device is ready. After a POR of the device, apply a CPU reset.

The CPU reset restarts the RBL and allows the NAND flash device to complete its initial transition out of reset. This works because NAND devices complete the second and subsequent reset sequences faster when compared to the initial reset sequence (5 µs versus 300 µs). NAND boot works by putting the C6657 or C6655 devices back into reset and then releasing it from reset. This extra reset sequence must be done without cycling power to the NAND device. Resetting the C6655 or C6657 devices after an initial NAND boot attempt works because the wait loop is long enough for success on a second boot attempt.

The C6657 EVM comes with a secondary boot loader that is written on the $I^2C$ EEPROM that is on the EVM. The FPGA firmware v02 forces the device to execute the IBL before it can execute any other code. This is a requirement for most of the out-of-box demos on the EVM.

Upgrading the firmware to v03 allows users to bypass the IBL and native boot from the boot media using the ROM code on the device. The FPGA bit files can be downloaded from EVM manufacturers resources download link.

## 11 Where can I download the KeyStone™ I EVM resources (user's guide, startup guide, schematics, BOM, FPGA bit file, and more)?

Use the following links to find related documents and files for each of the C6657, C6670, and C6678 devices.

- *TMS320C6657 - EVM*
- *TMS320C6670 Lite Evaluation Modules*
- *TMS320C6678 Lite Evaluation Modules*

## 12 I have programmed the IBL and want to boot the application from NOR flash. How do I convert a .out file to a .bin file to flash it?

You can re-name the .out file to .bin to program on NOR flash. If the file size is large then users can remove symbols from the binary using the strip6x utility. Use the strip6x.exe application to convert the .out file to a .bin format. See the following code snippets.

```
"strip6x.exe -p -o <Output file name .bin> <Input file
          .out>"
```

Example:

```
C:\ti\C6000 Code Generation Tools 7.4.0\bin>strip6x.exe -p -
o ..\..\..\Bin\nor.bin ..\..\..\Bin\hua_evmc6657l.out
```

## 13 Can the C6678 be booted up directly from SPI Nor Flash without the participation of I²C? Is it necessary to program IBL and IBL configuration on I²C EEPROM at bus address 0x51 when I test SPI boot mode on the TMDXEVM6678L?

Yes, it is possible. With the Silicon revision 1.0, the PLL fix is required. With the silicon revision 2.0, the PLL fix is not required (the IBL workaround is not required).

On the KeyStone I EVM for ROM boot modes (EMAC, SRIO, PCIe, Hyperlink, SPI, and more) and I²C boot mode with bus address 0x50, DSP initially boots from I²C EEPROM bus address 0x51 (IBL) that does the PLL reset workaround, updates the DEVSTAT for appropriate values based on the DIP switch settings (SW3 through SW6) and then re-enters the ROM to accomplish the desired boot mode.

Please review the errata document to understand the conditions of the PLL locking problem and the workaround; this is *Advisory 8*. The errata specifies a workaround option for no-boot, SPI, and I²C boot modes so users can do the same thing with a, SPI Flash instead of an I²C EEPROM.

On the EVM, this workaround is implemented using an FPGA and I²C EEPROM 0x51 to allow users to have as much flexibility as possible with the EVM boot selection. If the EVM is used, it is necessary to program the IBL and its configurations in address 0x51.

## 14 I am able to directly boot the SPI NOR example, but it fails to boot with complex .out (big .out file) file. How do you solve this issue?

If you are using the b2i2c utilities in the MCSDK, please look at the source for the B2i2c function in the ~/tools/ibl/src/util folder. The default max size configured in the tool is 0x20000. You can change the macro SIZE to be able to process your image (see the following code snippet).

```
#define SIZE 0x20000
```

## 15 After flashing IBL, I am trying to program the IBL configuration. After loading i2cConfig.gel, I cannot run the GEL script *EVM c6678 IBL → setConfig_c6678_main* and am stuck at this step. I have checked under the Scripts menu, but it has no dropdown menu shown. How do you solve this issue?

Follow the step-by-step procedure at the following link location to resolve this issue: *Where is script: "EVM c6678 IBL" in IBL Configuration?*.

## 16 How do I modify the IBL configuration table contents? For example, the IBL on the TMDSEVM66xx is not configured to boot an ELF image from NAND, and to do so I must change the IBL configuration table.

There are two methods to modify the IBL configuration table contents. Users can:

- Use the i2cConfig.gel file to update ibl_BOOT_FORMAT_BBLOB to ibl_BOOT_FORMAT_ELF in the i2cConfig.gel file. Refer to IBL configuration FAQ for updating the configuration table in I²C.

  or

- Modify the same file as the first option and then program the rebuilt IBL binary.

For example, edit the ~\tools\boot_loader\ibl\src\util\iblConfig\src\device.c file by changing ibl_t c6678_ibl_config(void) → if you are using the C6678.

```
ibl.bootModes[1].u.nandBoot.bootFormat        = ibl_BOOT_FORMAT_ELF;
```

## 17 When there are bad blocks in NAND flash, can the program in the C66xx boot from NAND flash normally?

Yes, because there is a bad block check implemented in the ROM boot loader. The ROM boot loader looks for a valid boot image in a sequential manner while skipping (and marking) any bad blocks (up to the first 32 blocks) on the NAND before reporting a boot failure.

## 18 How do you run the SRIO boot example using the C6670 and C6678 EVMs?

The SRIO boot example requires two EVMs and a breakout card to execute the example. The detailed procedure of setup and breakout card configuration are documented in the *DSP 6670 SRIO Boot loader example* forum.

## 19 How do you do an EMIF16 NOR Flash boot on the C66xx devices?

Please refer to the *C6678 EMIF16 boot summary* and *EMIF16 NOR Flash Booting of C6672 Custom board* forums.

## 20 How do you boot MAD image directly from RBL (without the EVM's IBL)?

Please refer to the following forum post for information: C6678 EVM How to boot MAD image directly from RBL (without the EVM's IBL).

## 21 How do you find the Silicon revision of the SoC mounted on the EVM or a custom board?

Refer to the *JTAG ID Register (JTAGID) Description* and *C66x DSP Device Nomenclature* sections of the data manual to find the Silicon revision.

## 22 I have used a different NAND chip in a custom board. Does the nandwrite.c file need to be modify, and how do you modify it?

The NANDWrite.c code provided with the example was built for the NAND flash used on the EVM and was created only to validate the boot during EVM functional validation. Users must modify the NAND writer code to match the NAND geometry provided in the datasheet of the NAND that you are using.

The geometry the user provided for the NAND does not seem to match the NAND geometry that is used on the EVM which is why the user may be encountering this issue. Change the following parameters in nandwrite.c to match the geometry of the NAND.

```
unsigned int busWidth = 8;
unsigned int pageSizeBytes = 2048;
unsigned int pagesPerBlock = 64;
unsigned int spareBytesPerSegment = 16;
unsigned int spareBytesPerPage = 0;
```

TI also provides a NANDWriter under the ~\tools\writer\nand\evmc66xxl\bin directory. Instructions to flash the NAND using the NANDWriter are provided in the ReadMe.txt file under the ~\tools\writer\nand\docs directory.

## 23 Can I use the MCSDK IBL for custom boards? Please describe the changes required for a custom IBL, if any.

Yes, users can use the IBL for custom boards, however, it must be ported for custom board configurations. The IBL provided in the MCSDK and Processor SDK package is provided for the EVMs that have an FPGA on it.

Users may need to modify the IBL if the custom board has different hook ups to boot media. For example, CS on which NAND is connected is different, NAND flash device geometry is different, change the clocking function based on the input clock etc.

There are few transactions made from the IBL code with the FPGA over the SPI interface. Please refer to the C66xxInit.c file in the IBL source code to look at the related FPGA initialization code. If the design does not contain an FPGA, users can clean up the code that sets up the FPGA configuration and should able to boot the IBL from the $I^2C$ EEPROM.

### *23.1 Building the IBL*

For build instructions and C66xx documentation, refer to the doc folder in the ~\tools\boot_loader\ibl\doc directory.

### *23.2 E2E™ Online Community References*

The following three forum posts on the TI E2E™ online community detail other build issues and solutions.

- *ibl for 6678 custom board*
- *MCSDK 2.01.02.06 IBL build errors*
- *rebuilding IBL for 6678*

## 24 Where can I find build and program instructions for KeyStone™ I Boot Examples and utilities?

The KeyStone I boot resources are documented as ReadMe.txt files in MCSDK SDK installation directories. Please refer to the following directory locations.

- Writers: ~\tools\writer\<eeprom/nand/nor>\docs
- POST: ~\tools\post\docs
- IBL: ~\tools\boot_loader\ibl\doc
- Boot Examples: ~\tools\boot_loader\examples\<ethernet/i2c/mad/pcie/srio>\docs

## 25 How do I load the EVMs with factory images?

Please follow the instructions from the program EVM user guide (.pdf) of the MCSDK and Processor SDK located in the following directory: ~\ti\mcsdk_2_0x_0x_0x\tools\program_evm.

## 26 How do I test, validate, and bring up the custom boards with C66xx devices?

Use the following instructions to test, validate, and bring up customer boards with C66xx devices.

1. Port the .gel file for the custom board with respect to the changes (such as clock, speed, memory configurations) by referring to and using TI provided utilities (such as clocking and DDR leveling spreadsheet, and more).
2. Port the Platform_test (or other diagnostics created for the EVM) for the custom board and test it.

There is a simple DDR test that is part of the .gel file called ddr3_memory_test (). Users can control the start and end address of the space.

## 27 Are there any validated big-endian boot examples for C66xx ?

No. By default, all of the boot examples are validated on little-endian mode only.

## 28 Where can I find the ROM bootloader (RBL) sources?

Please find the RBL sources for PG1.0 and 2.0 from the boot examples and reference code section of *Keystone Device Architecture*.

## 29 Would NOR code for EMIF be the same as NAND code? Is there any example for an EMIF16 NOR writer?

Refer to the following two forum posts on the TI E2E™ online community for a simple NOR writer.

- *Can I get EMIF16 sample code as NOR I/F on C6657?*
- *EMIF16 NOR TMS320C6678*

## IMPORTANT NOTICE

Texas Instruments Incorporated and its subsidiaries (TI) reserve the right to make corrections, enhancements, improvements and other changes to its semiconductor products and services per JESD46, latest issue, and to discontinue any product or service per JESD48, latest issue. Buyers should obtain the latest relevant information before placing orders and should verify that such information is current and complete. All semiconductor products (also referred to herein as “components”) are sold subject to TI’s terms and conditions of sale supplied at the time of order acknowledgment.

TI warrants performance of its components to the specifications applicable at the time of sale, in accordance with the warranty in TI’s terms and conditions of sale of semiconductor products. Testing and other quality control techniques are used to the extent TI deems necessary to support this warranty. Except where mandated by applicable law, testing of all parameters of each component is not necessarily performed.

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

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 relating to any combination, machine, or process in which TI components or services are used. Information published by TI regarding third-party products or services does not constitute a license 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 significant portions of TI 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. TI is not responsible or liable for such altered documentation. Information of third parties may be subject to additional restrictions.

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

Buyer acknowledges and agrees that it is solely responsible for compliance with all legal, regulatory and safety-related requirements concerning its products, and any use of TI components in its applications, notwithstanding any applications-related information or support that may be provided by TI. Buyer represents and agrees that it has all the necessary expertise to create and implement safeguards which anticipate dangerous consequences of failures, monitor failures and their consequences, lessen the likelihood of failures that might cause harm and take appropriate remedial actions. Buyer will fully indemnify TI and its representatives against any damages arising out of the use of any TI components in safety-critical applications.

In some cases, TI components may be promoted specifically to facilitate safety-related applications. With such components, TI’s goal is to help enable customers to design and create their own end-product solutions that meet applicable functional safety standards and requirements. Nonetheless, such components are subject to these terms.

No TI components are authorized for use in FDA Class III (or similar life-critical medical equipment) unless authorized officers of the parties have executed a special agreement specifically governing such use.

Only those TI components which TI has specifically designated as military grade or “enhanced plastic” are designed and intended for use in military/aerospace applications or environments. Buyer acknowledges and agrees that any military or aerospace use of TI components which have ***not*** been so designated is solely at the Buyer's risk, and that Buyer is solely responsible for compliance with all legal and regulatory requirements in connection with such use.

TI has specifically designated certain components as meeting ISO/TS16949 requirements, mainly for automotive use. In any case of use of non-designated products, TI will not be responsible for any failure to meet ISO/TS16949.

| **Products** | | **Applications** | |
|---|---|---|---|
| Audio | www.ti.com/audio | Automotive and Transportation | www.ti.com/automotive |
| Amplifiers | amplifier.ti.com | Communications and Telecom | www.ti.com/communications |
| Data Converters | dataconverter.ti.com | Computers and Peripherals | www.ti.com/computers |
| DLP® Products | www.dlp.com | Consumer Electronics | www.ti.com/consumer-apps |
| DSP | dsp.ti.com | Energy and Lighting | www.ti.com/energy |
| Clocks and Timers | www.ti.com/clocks | Industrial | www.ti.com/industrial |
| Interface | interface.ti.com | Medical | www.ti.com/medical |
| Logic | logic.ti.com | Security | www.ti.com/security |
| Power Mgmt | power.ti.com | Space, Avionics and Defense | www.ti.com/space-avionics-defense |
| Microcontrollers | microcontroller.ti.com | Video and Imaging | www.ti.com/video |
| RFID | www.ti-rfid.com | | |
| OMAP Applications Processors | www.ti.com/omap | **TI E2E Community** | e2e.ti.com |
| Wireless Connectivity | www.ti.com/wirelessconnectivity | | |

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