EK-LM4F232 Firmware Development Package

USER'S GUIDE

SW-EK-LM4F232-UG-2.1.4.178

Copyright

Texas Instruments
108 Wild Basin, Suite 350
Austin, TX 78746
www.ti.com/tiva-c

Revision Information

This is version 2.1.4.178 of this document, last updated on February 22, 2017.

Table of Contents

1 Introduction

The Texas Instruments® Stellaris® EK-LM4F232 evaluation board is a platform that can be used for software development and prototyping a hardware design. It can also be used as a guide for custom board design using a Stellaris microcontroller.

The EK-LM4F232 includes a Stellaris ARM® Cortex™-M3-based microcontroller and the following features:

- Stellaris® LM4F232H5QD microcontroller
- Four 20V analog inputs
- 3-axis analog accelerometer
- On-board temperature sensor
- Bright 96 x 64 16-bit color OLED display
- 5 user buttons
- User LED
- Shunt for microcontroller current consumption measurement
- MicroSD card connector
- USB OTG connector
- On-board In-Circuit Debug Interface (ICDI)
- Coin cell backup battery for Hibernate feature
- Power supply option from USB ICDI connection, or OTG connection

This document describes the board-specific drivers and example applications that are provided for this development board.

2 Example Applications

The example applications show how to utilize features of the EK-LM4F232 evaluation board. Examples are included to show how to use many of the general features of the Stellaris microcontroller, as well as the feature that are unique to this evaluation board.

A number of drivers are provided to make it easier to use the features of the EK-LM4F232. These drivers also contain low-level code that make use of the Stellaris peripheral driver library and utilities.

There is an IAR workspace file (`ek-lm4f232.eww`) that contains the peripheral driver library project, along with all of the board example projects, in a single, easy-to-use workspace for use with Embedded Workbench version 5.

There is a Keil multi-project workspace file (`ek-lm4f232.uvmpw`) that contains the peripheral driver library project, along with all of the board example projects, in a single, easy-to-use workspace for use with uVision.

All of these examples reside in the `examples/boards/ek-lm4f232` subdirectory of the firmware development package source distribution.

2.1 Bit-Banding (bitband)

This example application demonstrates the use of the bit-banding capabilities of the Cortex-M3 microprocessor. All of SRAM and all of the peripherals reside within bit-band regions, meaning that bit-banding operations can be applied to any of them. In this example, a variable in SRAM is set to a particular value one bit at a time using bit-banding operations (it would be more efficient to do a single non-bit-banded write; this simply demonstrates the operation of bit-banding).

2.2 Blinky (blinky)

A very simple example that blinks the on-board LED.

2.3 Boot Loader Demo 1 (boot_demo1)

An example to demonstrate the use of a flash-based boot loader. At startup, the application will configure the UART and USB peripherals, and then branch to the boot loader to await the start of an update. If using the serial boot loader (boot_serial), the UART will always be configured at 115,200 baud and does not require the use of auto-bauding.

This application is intended for use with any of the three flash-based boot loader flavors (boot_serial or boot_usb) included in the software release. To accommodate the largest of these (boot_usb), the link address is set to 0x2800. If you are using serial, you may change this address to a 1KB boundary higher than the last address occupied by the boot loader binary as long as you also rebuild the boot loader itself after modifying its bl_config.h file to set APP_START_ADDRESS to the same value.

The boot_demo2 application can be used along with this application to easily demonstrate that the boot loader is actually updating the on-chip flash.

Note that the LM4F232 and other Blizzard-class devices also support the serial and USB boot loaders in ROM. To make use of this function, link your application to run at address 0x0000 in flash and enter the bootloader using the ROM_UpdateSerial and ROM_UpdateUSB functions (defined in rom.h). This mechanism is used in the utils/swupdate.c module when built specifically targeting a suitable Blizzard-class device.

2.4 Boot Loader Demo 2 (boot_demo2)

An example to demonstrate the use of a flash-based boot loader. At startup, the application will configure the UART, USB and Ethernet peripherals, wait for a widget on the screen to be pressed, and then branch to the boot loader to await the start of an update. If using the serial boot loader (boot_serial), the UART will always be configured at 115,200 baud and does not require the use of auto-bauding.

This application is intended for use with any of the three flash-based boot loader flavors (boot_eth, boot_serial or boot_usb) included in the software release. To accommodate the largest of these (boot_usb), the link address is set to 0x2800. If you are using serial or Ethernet boot loader, you may change this address to a 1KB boundary higher than the last address occupied by the boot loader binary as long as you also rebuild the boot loader itself after modifying its bl_config.h file to set APP_START_ADDRESS to the same value.

The boot_demo1 application can be used along with this application to easily demonstrate that the boot loader is actually updating the on-chip flash.

Note that the LM4F232 and other Blizzard-class devices also the support serial and USB boot loaders in ROM. To make use of this function, link your application to run at address 0x0000 in flash and enter the bootloader using either the ROM_UpdateUSB or ROM_UpdateSerial functions (defined in rom.h). This mechanism is used in the utils/swupdate.c module when built specifically targeting a suitable Blizzard-class device.

2.5 Boot Loader (boot_serial)

The boot loader is a small piece of code that can be programmed at the beginning of flash to act as an application loader as well as an update mechanism for an application running on a Tiva C Series microcontroller, utilizing either UART0, I2C0, SSI0, or USB. The capabilities of the boot loader are configured via the bl_config.h include file. For this example, the boot loader uses UART0 to load an application.

The configuration is set to boot applications which are linked to run from address 0x2800 in flash. This is higher than strictly necessary but is intended to allow the example boot loader-aware applications provided in the release to be used with any of the three boot loader example configurations supplied (serial or USB) without having to adjust their link addresses.

Note that the LM4F232 and other Blizzard-class devices also support serial boot loaders in ROM.

2.6 USB Boot Loader (boot_usb)

The boot loader is a small piece of code that can be programmed at the beginning of flash to act as an application loader as well as an update mechanism for an application running on a Tiva C Series microcontroller, utilizing either UART0, I2C0, SSI0, or USB. The capabilities of the boot loader are configured via the bl_config.h include file. For this example, the boot loader uses the USB Device Firmware Upgrade (DFU) class to download an application.

Applications intended for use with this version of the boot loader should be linked to run from address 0x2800 in flash (rather than the default run address of 0). This address is chosen to ensure that boot loader images built with all supported compilers may be used without modifying the application start address. Depending upon the compiler and optimization level you are using, however, you may find that you can reclaim some space by lowering this address and rebuilding both the application and boot loader. To do this, modify the makefile or project you use to build the application to show the new run address and also change the APP_START_ADDRESS value defined in bl_config.h before rebuilding the boot loader.

The USB boot loader may be demonstrated using the boot_demo1 and boot_demo2 example applications in addition to the boot_usb boot loader binary itself. Note that these are the only two example applications currently configured to run alongside the USB boot loader but making any of the other applications boot loader compatible is simply a matter of relinking them with the new start address and adding a mechanism to transfer control to the boot loader when required.

The Windows device driver required to communicate with the USB boot loader can be found on the software and documentation CD from the development kit package. It can also be found in the Windows driver package which can be downloaded via a link from `http://www.ti.com/tiva`.

A Windows command-line application, dfuprog, is also provided which illustrates how to perform uploads and downloads via the USB DFU protocol. The source for this application can be found in the /ti/TivaWare_C_Series-x.x/tools directory and the prebuilt executable is available in the package “TivaWare for C Series PC Companion Utilities” available for download via a link from `http://www.ti.com/tivaware`.

2.7 Hello World (hello)

A very simple “hello world” example. It simply displays “Hello World!” on the display and is a starting point for more complicated applications. This example uses calls to the TivaWare Graphics Library graphics primitives functions to update the display. For a similar example using widgets, please see “hello_widget”.

2.8 Hibernate Example (hibernate)

An example to demonstrate the use of the Hibernation module. The user can put the microcontroller in hibernation by pressing the select button. The microcontroller will then wake on its own after 5 seconds, or immediately if the user presses the select button again. The program keeps a count of the number of times it has entered hibernation. The value of the counter is stored in the battery backed memory of the Hibernation module so that it can be retrieved when the microcontroller wakes.

2.9 Interrupts (interrupts)

This example application demonstrates the interrupt preemption and tail-chaining capabilities of Cortex-M4 microprocessor and NVIC. Nested interrupts are synthesized when the interrupts have the same priority, increasing priorities, and decreasing priorities. With increasing priorities, preemption will occur; in the other two cases tail-chaining will occur. The currently pending interrupts and the currently executing interrupt will be displayed on the display; GPIO pins D0, D1 and D2 will be asserted upon interrupt handler entry and de-asserted before interrupt handler exit so that the off-to-on time can be observed with a scope or logic analyzer to see the speed of tail-chaining (for the two cases where tail-chaining is occurring).

2.10 MPU (mpu_fault)

This example application demonstrates the use of the MPU to protect a region of memory from access, and to generate a memory management fault when there is an access violation.

2.11 Data Logger (qs-logger)

This example application is a data logger. It can be configured to collect data from up to 10 data sources. The possible data sources are:

- 4 analog inputs, 0-20V
- 3-axis accelerometer
- internal and external temperature sensors
- processor current consumption

The data logger provides a menu navigation that is operated by the buttons on the EK-LM4F232 board (up, down, left, right, select). The data logger can be configured by using the menus. The following items can be configured:

- data sources to be logged
- sample rate
- storage location
- sleep modes
- clock

Using the data logger:

Use the CONFIG menu to configure the data logger. The following choices are provided:

- CHANNELS - enable specific channels of data that will be logged
- PERIOD - select the sample period
- STORAGE - select where the collected data will be stored:
  - FLASH - stored in the internal flash memory
  - USB - stored on a connected USB memory stick

  - HOST PC - transmitted to a host PC via USB OTG virtual serial port
  - NONE - the data will only be displayed and not stored
- SLEEP - select whether or not the board sleeps between samples. Sleep mode is allowed when storing to flash at with a period of 1 second or longer.
- CLOCK - allows setting of internal time-of-day clock that is used for time stamping of the sampled data

Use the START menu to start the data logger running. It will begin collecting and storing the data. It will continue to collect data until stopped by pressing the left button or select button.

While the data logger is collecting data and it is not configured to sleep, a simple strip chart showing the collected data will appear on the display. If the data logger is configured to sleep, then no strip chart will be shown.

If the data logger is storing to internal flash memory, it will overwrite the oldest data. If storing to a USB memory device it will store data until the device is full.

The VIEW menu allows viewing the values of the data sources in numerical format. When viewed this way the data is not stored.

The SAVE menu allows saving data that was stored in internal flash memory to a USB stick. The data will be saved in a text file in CSV format.

The ERASE menu is used to erase the internal memory so more data can be saved.

When the EK-LM4F232 board running qs-logger is connected to a host PC via the USB OTG connection for the first time, Windows will prompt for a device driver for the board. This can be found in /ti/TivaWare_C_Series-x.x/windows_drivers assuming you installed the software in the default folder.

A companion Windows application, logger, can be found in the /ti/TivaWare_C_Series-x.x/tools/bin directory. When the data logger's STORAGE option is set to "HOST PC" and the board is connected to a PC via the USB OTG connection, captured data will be transfered back to the PC using the virtual serial port that the EK board offers. When the logger application is run, it will search for the first connected EK-LM4F232 board and display any sample data received. The application also offers the option to log the data to a file on the PC.

2.12 SD card using FAT file system (sd_card)

This example application demonstrates reading a file system from an SD card. It makes use of FatFs, a FAT file system driver. It provides a simple command console via a serial port for issuing commands to view and navigate the file system on the SD card.

The first UART, which is connected to the USB debug virtual serial port on the evaluation board, is configured for 115,200 bits per second, and 8-N-1 mode. When the program is started a message will be printed to the terminal. Type “help” for command help.

For additional details about FatFs, see the following site: `http://elm-chan.org/fsw/ff/00index_e.html`

2.13 Sine Demo (sine_demo)

This example uses the floating point capabilities of the Tiva C Series processor to compute a sine wave and show it on the display.

2.14 SoftUART Echo (softuart_echo)

This example application utilizes the SoftUART to echo text. The SoftUART is configured to use the same pins as the first UART (connected to the FTDI virtual serial port on the evaluation board), at 115,200 baud, 8-n-1 mode. All characters received on the SoftUART are transmitted back to the SoftUART.

2.15 Timer (timers)

This example application demonstrates the use of the timers to generate periodic interrupts. One timer is set up to interrupt once per second and the other to interrupt twice per second; each interrupt handler will toggle its own indicator on the display.

2.16 UART Echo (uart_echo)

This example application utilizes the UART to echo text. The first UART (connected to the USB debug virtual serial port on the evaluation board) will be configured in 115,200 baud, 8-n-1 mode. All characters received on the UART are transmitted back to the UART.

2.17 uDMA (udma_demo)

This example application demonstrates the use of the uDMA controller to transfer data between memory buffers, and to transfer data to and from a UART. The test runs for 10 seconds before exiting.

2.18 USB Generic Bulk Device (usb_dev_bulk)

This example provides a generic USB device offering simple bulk data transfer to and from the host. The device uses a vendor-specific class ID and supports a single bulk IN endpoint and a single bulk OUT endpoint. Data received from the host is assumed to be ASCII text and it is echoed back with the case of all alphabetic characters swapped.

A Windows INF file for the device is provided on the installation CD and in the C:/ti/TivaWare_C_Series-x.x/windows_drivers directory of TivaWare releases. This INF contains information required to install the WinUSB subsystem on WindowsXP and Vista PCs. WinUSB is a

Windows subsystem allowing user mode applications to access the USB device without the need for a vendor-specific kernel mode driver.

A sample Windows command-line application, usb_bulk_example, illustrating how to connect to and communicate with the bulk device is also provided. The application binary is installed as part of the "TivaWare for C Series PC Companion Utilities" package (SW-TM4C-USB-WIN) on the installation CD or via download from `http://www.ti.com/tivaware`. Project files are included to allow the examples to be built using Microsoft Visual Studio 2008. Source code for this application can be found in directory ti/TivaWare_C_Series-x.x/tools/usb_bulk_example.

2.19 USB HID Keyboard Device (usb_dev_keyboard)

This example application turns the evaluation board into a USB keyboard supporting the Human Interface Device class. When the push button is pressed, a sequence of key presses is simulated to type a string. Care should be taken to ensure that the active window can safely receive the text; enter is not pressed at any point so no actions are attempted by the host if a terminal window is used (for example). The status LED is used to indicate the current Caps Lock state and is updated in response to any other keyboard attached to the same USB host system.

The device implemented by this application also supports USB remote wakeup allowing it to request the host to reactivate a suspended bus. If the bus is suspended (as indicated on the application display), pressing the push button will request a remote wakeup assuming the host has not specifically disabled such requests.

2.20 USB MSC Device (usb_dev_msc)

This example application turns the evaluation board into a USB mass storage class device. The application will use the microSD card for the storage media for the mass storage device. The screen will display the current action occurring on the device ranging from disconnected, no media, reading, writing and idle.

2.21 USB Serial Device (usb_dev_serial)

This example application turns the evaluation kit into a virtual serial port when connected to the USB host system. The application supports the USB Communication Device Class, Abstract Control Model to redirect UART0 traffic to and from the USB host system.

Assuming you installed TivaWare in the default directory, a driver information (INF) file for use with Windows XP, Windows Vista and Windows7 can be found in C:/ti/TivaWare_C_Series-x.x/windows_drivers. For Windows 2000, the required INF file is in C:/ti/TivaWare_C_Series-x.x/windows_drivers/win2K.

2.22 USB host audio example application using SD Card FAT file system (usb_host_audio)

This example application demonstrates playing .wav files from an SD card that is formatted with a FAT file system using USB host audio class. The application will only look in the root directory of the SD card and display all files that are found. Files can be selected to show their format and then played if the application determines that they are a valid .wav file. Only PCM format (uncompressed) files may be played.

For additional details about FatFs, see the following site: `http://elm-chan.org/fsw/ff/00index_e.html`

2.23 USB HID Keyboard Host (usb_host_keyboard)

This example application demonstrates how to support a USB keyboard attached to the evaluation kit board. The display will show if a keyboard is currently connected and the current state of the Caps Lock key on the keyboard that is connected on the bottom status area of the screen. Pressing any keys on the keyboard will cause them to be printed on the screen and to be sent out the UART at 115200 baud with no parity, 8 bits and 1 stop bit. Any keyboard that supports the USB HID BIOS protocol should work with this demo application.

2.24 USB HID Mouse Host (usb_host_mouse)

This application demonstrates the handling of a USB mouse attached to the evaluation kit. Once attached, the position of the mouse pointer and the state of the mouse buttons are output to the display.

2.25 USB Mass Storage Class Host Example (usb_host_msc)

This example application demonstrates reading a file system from a USB flash disk. It makes use of FatFs, a FAT file system driver. It provides a simple widget-based display for showing and navigating the file system on a USB stick.

For additional details about FatFs, see the following site: `http://elm-chan.org/fsw/ff/00index_e.html`

2.26 USB Stick Update Demo (usb_stick_demo)

An example to demonstrate the use of the flash-based USB stick update program. This example is meant to be loaded into flash memory from a USB memory stick, using the USB stick update program (usb_stick_update), running on the microcontroller.

After this program is built, the binary file (usb_stick_demo.bin), should be renamed to the filename expected by usb_stick_update ("FIRMWARE.BIN" by default) and copied to the root directory of a USB memory stick. Then, when the memory stick is plugged into the eval board that is running the usb_stick_update program, this example program will be loaded into flash and then run on the microcontroller.

This program simply displays a message on the screen and prompts the user to press the select button. Once the button is pressed, control is passed back to the usb_stick_update program which is still is flash, and it will attempt to load another program from the memory stick. This shows how a user application can force a new firmware update from the memory stick.

2.27 USB Memory Stick Updater (usb_stick_update)

This example application behaves the same way as a boot loader. It resides at the beginning of flash, and will read a binary file from a USB memory stick and program it into another location in flash. Once the user application has been programmed into flash, this program will always start the user application until requested to load a new application.

When this application starts, if there is a user application already in flash (at **APP_START_ADDRESS**), then it will just run the user application. It will attempt to load a new application from a USB memory stick under the following conditions:

- no user application is present at **APP_START_ADDRESS**
- the user application has requested an update by transferring control to the updater
- the user holds down the eval board push button when the board is reset

When this application is attempting to perform an update, it will wait forever for a USB memory stick to be plugged in. Once a USB memory stick is found, it will search the root directory for a specific file name, which is *FIRMWARE.BIN* by default. This file must be a binary image of the program you want to load (the .bin file), linked to run from the correct address, at **APP_START_ADDRESS**.

The USB memory stick must be formatted as a FAT16 or FAT32 file system (the normal case), and the binary file must be located in the root directory. Other files can exist on the memory stick but they will be ignored.

2.28 Watchdog (watchdog)

This example application demonstrates the use of the watchdog as a simple heartbeat for the system. If the watchdog is not periodically fed, it will reset the system. Each time the watchdog is fed, the LED is inverted so that it is easy to see that it is being fed, which occurs once every second. To stop the watchdog being fed and, hence, cause a system reset, press the select button.

3 Buttons Driver

3.1 Introduction

The buttons driver provides functions to make it easy to use the push buttons on the EK-LM4F232 evaluation board. The driver provides a function to initialize all the hardware required for the buttons, and features for debouncing and querying the button state.

This driver is located in `examples/boards/ek-lm4f232/drivers`, with `buttons.c` containing the source code and `buttons.h` containing the API declarations for use by applications.

3.2 API Functions

Functions

- void ButtonsInit (void)
- uint8_t ButtonsPoll (uint8_t *pui8Delta, uint8_t *pui8RawState)

3.2.1 Function Documentation

3.2.1.1 ButtonsInit

Initializes the GPIO pins used by the board pushbuttons.

Prototype:

```
void
ButtonsInit(void)
```

Description:

This function must be called during application initialization to configure the GPIO pins to which the pushbuttons are attached. It enables the port used by the buttons and configures each button GPIO as an input with a weak pull-up.

Returns:

None.

3.2.1.2 ButtonsPoll

Polls the current state of the buttons and determines which have changed.

Prototype:

```
uint8_t
ButtonsPoll(uint8_t *pui8Delta,
            uint8_t *pui8RawState)
```

Parameters:

pui8Delta points to a character that will be written to indicate which button states changed since the last time this function was called. This value is derived from the debounced state of the buttons.

pui8RawState points to a location where the raw button state will be stored.

Description:

This function should be called periodically by the application to poll the pushbuttons. It determines both the current debounced state of the buttons and also which buttons have changed state since the last time the function was called.

In order for button debouncing to work properly, this function should be caled at a regular interval, even if the state of the buttons is not needed that often.

If button debouncing is not required, the the caller can pass a pointer for the *pui8RawState* parameter in order to get the raw state of the buttons. The value returned in *pui8RawState* will be a bit mask where a 1 indicates the buttons is pressed.

Returns:

Returns the current debounced state of the buttons where a 1 in the button ID's position indicates that the button is pressed and a 0 indicates that it is released.

3.3 Programming Example

The following example shows how to use the buttons driver to initialize the buttons, debounce and read the buttons state.

```
//
// Initialize the buttons.
//
ButtonsInit();

//
// From timed processing loop (for example every 10 ms)
//
...
{
    //
    // Poll the buttons.  When called periodically this function will
    // run the button debouncing algorithm.
    //
    ucState = ButtonsPoll(&ucDelta, 0);

    //
    // Test to see if the SELECT button was pressed and do something
    //
    if(BUTTON_PRESSED(SELECT_BUTTON, ucState, ucDelta))
    {
        ...
        // SELECT button action
    }
}
```

4 Display Driver

4.1 Introduction

The display driver offers a standard interface to access display functions on the CrystalFontz 96x64 16-bit color OLED display and is used by the Stellaris Graphics Library and widget manager. In addition to providing the `tDisplay` structure required by the graphics library, the display driver also provides an API for initializing the display.

This driver is located in `examples/boards/ek-lm4f232/drivers`, with `cfal96x64x16.c` containing the source code and `cfal96x64x16.h` containing the API declarations for use by applications.

4.2 API Functions

Functions

- void CFAL96x64x16Init (void)

Variables

- const tDisplay g_sCFAL96x64x16

4.2.1 Function Documentation

4.2.1.1 CFAL96x64x16Init

Initializes the display driver.

Prototype:

```
void
CFAL96x64x16Init(void)
```

Description:

This function initializes the SSD1332 display controller on the panel, preparing it to display data.

Returns:

None.

4.2.2 Variable Documentation

4.2.2.1 g_sCFAL96x64x16

Definition:

```
const tDisplay g_sCFAL96x64x16
```

Description:

The display structure that describes the driver for the Crystalfontz CFAL9664-F-B1 OLED panel with SSD 1332 controller.

4.3 Programming Example

The following example shows how to initialize the display and prepare to draw on it using the graphics library.

```
tContext sContext;

//
// Initialize the display.
//
CFAL96x64x16Init();

//
// Initialize a graphics library drawing context.
//
GrContextInit(&sContext, &g_sCFAL96x64x16);
```

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