EK-TM4C123GXL Firmware Development Package

USER'S GUIDE

SW-EK-TM4C123GXL-UG-2.2.0.295

Copyright

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

Revision Information

This is version 2.2.0.295 of this document, last updated on April 02, 2020.

Table of Contents

1 Introduction

The Texas Instruments® Tiva™ C Series EK-TM4C123GXL evaluation board is a low cost platform that can be used for software development and to prototype a hardware design. It contains a Tiva C Series ARM® Cortex™-M4F-based microcontroller, a USB device port, two push buttons, and a RGB LED that can be used to exercise the peripherals on the microcontroller. Additionally, most of the microcontroller's pins are brought to headers, allowing for easy connection to other hardware for the purposes of prototyping. The outer rows of header pins are compatible with the MSP430™ Launchpad.

This document describes the example applications that are provided for this evaluation board.

2 Example Applications

The example applications show how to use features of the Cortex-M4F microprocessor, the peripherals on the Tiva C Series microcontroller, and the drivers provided by the peripheral driver library. These applications are intended for demonstration and as a starting point for new applications.

There is an IAR workspace file (`ek-tm4c123gxl.eww`) that contains the peripheral driver library project, USB library project, and all of the board example projects, in a single, easy to use workspace for use with Embedded Workbench version 6.

There is a Keil multi-project workspace file (`ek-tm4c123gxl.mpw`) that contains the peripheral driver library project, USB library project, and 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-tm4c123gxl` subdirectory of the firmware development package source distribution.

2.1 ADC with Timer Trigger and uDMA transfer Demo (adc_udma_pingpong)

This example demonstrates how to use the ADC peripheral with both the uDMA and Timer peripherals to optimize the ADC sampling process. The uDMA is configured for Ping-Pong mode and used to transfer ADC measurement results into a buffer in the background to minimize CPU usage and then indicate when the buffer is ready for processing by the application. The Timer is used to trigger the ADC measurements at a set sampling frequency of 16 kHz.

This example uses the following peripherals and I/O signals. You must review these and change as needed for your own board:

- ADC0 peripheral
- GPIO Port E peripheral (for AIN0 pin)
- AIN0 - PE0

UART0, connected to the Virtual Serial Port and running at 115,200, 8-N-1, is used to display messages from this application.

2.2 Bit-Banding (bitband)

This example application demonstrates the use of the bit-banding capabilities of the Cortex-M4F 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).

2.3 Blinky (blinky)

A very simple example that blinks the on-board LED using direct register access.

2.4 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 two 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 TM4C123G 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.

2.5 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 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 two 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_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 TM4C123G 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 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.

2.6 ROM UART Boot Loader Demo (boot_demo_uart_rom)

An example to demonstrate the use of a ROM-based boot loader. At startup, the application will configure the UART peripheral, and then branch to the ROM boot loader to await the start of an update. The UART will always be configured at 115,200 baud and does not require the use of auto-bauding.

Note: The newly loaded application should start from APP_BASE 0x00000000 and will override this boot loader example. This is the intended use of this example. For a persistent Flash-based boot loader, see the boot_serial example.

2.7 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 TM4C123G and other Blizzard-class devices also support serial boot loaders in ROM.

2.8 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_demo_usb example application.

The Windows device driver required to communicate with the USB boot loader can be found in `C:/TI/TivaWare_C_Series-x.x/windows_drivers` assuming you installed TivaWare in the default directory.

To illustrate runtime DFU capability, use the `dfuprog` tool which is part of the Tiva Windows USB Examples package (SW-USB-win-xxxx.msi) Assuming this package is installed in the default location, the `dfuprog` executable can be found in either the `C:/Program Files/Texas Instruments/Tiva/usb_examples` or `C:/Program Files (x86)/Texas Instruments/Tiva/usb_examples` directory.

The `dfuprog` tool is a Windows command-line application which illustrates how to perform uploads and downloads via the USB DFU protocol.

2.9 GPIO JTAG Recovery (gpio_jtag)

This example demonstrates changing the JTAG pins into GPIOs, aint32_t with a mechanism to revert them to JTAG pins. When first run, the pins remain in JTAG mode. Pressing the left button will toggle the pins between JTAG mode and GPIO mode. Because there is no debouncing of the push button (either in hardware or software), a button press will occasionally result in more than one mode change.

In this example, four pins (PC0, PC1, PC2, and PC3) are switched.

UART0, connected to the ICDI virtual COM port and running at 115,200, 8-N-1, is used to display messages from this application.

2.10 Hello World (hello)

A very simple “hello world” example. It simply displays “Hello World!” on the UART and is a starting point for more complicated applications.

UART0, connected to the Virtual Serial Port and running at 115,200, 8-N-1, is used to display messages from this application.

2.11 Hiberate Demo (hibernate)

This example demonstrates the use of the Hibernation module. The module will be configured for Real-Time Counter (RTC) mode using the 32.768kHz crystal that is installed on the EK-TM4C123GXL LaunchPad. With RTC mode, a match time will be loaded to trigger an interrupt that will wake the device from hibernation mode.

This example also uses a UART configured for 115200 baud, 8-N-1 mode to send the current RTC count on each interrupt.

2.12 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 E1, E2 and E3 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).

2.13 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.

UART0, connected to the virtual serial port and running at 115,200, 8-N-1, is used to display messages from this application.

2.14 Project Zero (project0)

This example demonstrates the use of TivaWare to setup the clocks and toggle GPIO pins to make the LED's blink. This is a good place to start understanding your launchpad and the tools that can be used to program it.

2.15 PWM Reload Interrupt Demo (pwm_interrupt)

This example shows how to configure PWM0 for a load interrupt. The PWM interrupt will trigger every time the PWM0 counter gets reloaded. In the interrupt, 0.1% will be added to the current duty cycle. This will continue until a duty cycle of 75% is received, then the duty cycle will get reset to 0.1%.

This example uses the following peripherals and I/O signals.

- GPIO Port B peripheral (for PWM0 pin)
- PWM0 - PB6

UART0, connected to the Virtual Serial Port and running at 115,200, 8-N-1, is used to display messages from this application.

2.16 EK-TM4C123GXL Quickstart Application (qs-rgb)

A demonstration of the Tiva C Series LaunchPad (EK-TM4C123GXL) capabilities.

Press and/or hold the left button to traverse towards the red end of the ROYGBIV color spectrum. Press and/or hold the right button to traverse toward the violet end of the ROYGBIV color spectrum.

If no input is received for 5 seconds, the application will start automatically changing the color displayed.

Press and hold both left and right buttons for 3 seconds to enter hibernation. During hibernation, the last color displayed will blink for 0.5 seconds every 3 seconds.

The system can also be controlled via a command line provided via the UART. Configure your host terminal emulator for 115200, 8-N-1 to access this feature.

- Command ’help’ generates a list of commands and helpful information.
- Command ’hib’ will place the device into hibernation mode.
- Command ’rand’ will initiate the pseudo-random color sequence.
- Command ’intensity’ followed by a number between 0 and 100 will set the brightness of the LED as a percentage of maximum brightness.
- Command ’rgb’ followed by a six character hex value will set the color. For example ’rgb FF0000’ will produce a red color.

2.17 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.

UART0, connected to the Virtual Serial Port and running at 115,200, 8-N-1, is used to display messages from this application.

2.18 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.

2.19 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.

UART0, connected to the FTDI virtual COM port and running at 115,200, 8-N-1, is used to display messages from this application.

2.20 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.

Assuming you installed TivaWare in the default directory, a driver information (INF) file for use with Windows XP, Windows Vista, Windows 7, and Windows 10 can be found in C:/TivaWare_C_Series-x.x/windows_drivers. For Windows 2000, the required INF file is in C:/TivaWare_C_Series-x.x/windows_drivers/win2K.

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 "Windows-side examples for USB kits" package (SW-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 VisualStudio 2008. Source code for this application can be found in directory TivaWare-for-C-Series/tools/usb_bulk_example.

2.21 USB HID Gamepad Device (usb_dev_gamepad)

This example application turns the evaluation board into USB game pad device using the Human Interface Device gamepad class. The buttons on the board are reported as buttons 1 and 2. The X, Y, and Z coordinates are reported using the ADC input on GPIO port E pins 1, 2, and 3. The X input is on PE3, the Y input is on PE2 and the Z input is on PE1. These are not connected to any real input so the values simply read whatever is on the pins. To get valid values the pins should have voltage that range from VDDA(3V) to 0V. The blue LED on PF5 is used to indicate gamepad activity to the host and blinks when there is USB bus activity.

2.22 USB HID Keyboard Device (usb_dev_keyboard)

This example turns the EK-TM4C123GXL LaunchPad into a USB keyboard supporting the Human Interface Device class. When either the SW1/SW2 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.

2.23 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, Windows 7, and Windows 10 can be found in C:/TivaWare_C_Series-x.x/windows_drivers. For Windows 2000, the required INF file is in C:/TivaWare_C_Series-x.x/windows_drivers/win2K.

2.24 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 cause a system reset, press the SW1 button.

UART0, connected to the Virtual Serial Port and running at 115,200, 8-N-1, is used to display messages from this application.

3 Buttons Driver

3.1 Introduction

The buttons driver provides functions to make it easy to use the push buttons on the EK-TM4C123GXL 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-tm4c123gxl/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 RGB LED Driver

4.1 Introduction

The RGB LED driver provides a simple interface to control the RGB LED on the EK-TM4C123GXL. The driver provides a function to initialize the timers and GPIO for the RGB. It also provides features for controlling the color and intensity of the LED.

This driver is located in `examples/boards/ek-tm4c123gxl/drivers`, with `rgb.c` containing the source code and `rgb.h` containing the API declarations for use by applications.

4.2 API Functions

Functions

- void RGBBlinkIntHandler (void)
- void RGBBlinkRateSet (float fRate)
- void RGBColorGet (uint32_t *pui32RGBColor)
- void RGBColorSet (volatile uint32_t *pui32RGBColor)
- void RGBDisable (void)
- void RGBEnable (void)
- void RGBInit (uint32_t ui32Enable)
- void RGBIntensitySet (float fIntensity)
- void RGBSet (volatile uint32_t *pui32RGBColor, float fIntensity)

4.2.1 Function Documentation

4.2.1.1 RGBBlinkIntHandler

Wide Timer interrupt to handle blinking effect of the RGB

Prototype:

```
void
RGBBlinkIntHandler(void)
```

Description:

This function is called by the hardware interrupt controller on a timeout of the wide timer. This function must be in the NVIC table in the startup file. When called will toggle the enable flag to turn on or off the entire RGB unit. This creates a blinking effect. A wide timer is used since the blink is intended to be visible to the human eye and thus is expected to have a frequency between 15 and 0.1 hz. Currently blink duty is fixed at 50%.

Returns:
None.

4.2.1.2 RGBBlinkRateSet

Sets the blink rate of the RGB Led

Prototype:
```
void
RGBBlinkRateSet(float fRate)
```

Parameters:
fRate is the blink rate in hertz.

Description:
This function controls the blink rate of the RGB LED in auto blink mode. to enable blinking pass a non-zero floating pointer number. To disable pass 0.0f as the argument. Calling this function will override the current RGBDisable or RGBEnable status.

Returns:
None.

4.2.1.3 RGBColorGet

Get the output color.

Prototype:
```
void
RGBColorGet(uint32_t *pui32RGBColor)
```

Parameters:
pui32RGBColor points to a three element array representing the relative intensity of each color. Red is element 0, Green is element 1, Blue is element 2. 0x0000 is off. 0xFFFF is fully on. Caller must allocate and pass a pointer to a three element array of uint32_ts.

Description:
This function should be called by the application to get the current color of the RGB LED.

Returns:
None.

4.2.1.4 RGBColorSet

Set the output color.

Prototype:
```
void
RGBColorSet(volatile uint32_t *pui32RGBColor)
```

Parameters:

pui32RGBColor points to a three element array representing the relative intensity of each color. Red is element 0, Green is element 1, Blue is element 2. 0x0000 is off. 0xFFFF is fully on.

Description:

This function should be called by the application to set the color of the RGB LED.

Returns:

None.

4.2.1.5 RGBDisable

Disable the RGB LED by configuring the GPIO's as inputs.

Prototype:

```
void
RGBDisable(void)
```

Description:

This function or RGBEnable should be called during application initialization to configure the GPIO pins to which the LEDs are attached. This function disables the timers and configures the GPIO pins as inputs for minimum current draw.

Returns:

None.

4.2.1.6 RGBEnable

Enable the RGB LED with already configured timer settings

Prototype:

```
void
RGBEnable(void)
```

Description:

This function or RGBDisable should be called during application initialization to configure the GPIO pins to which the LEDs are attached. This function enables the timers and configures the GPIO pins as timer outputs.

Returns:

None.

4.2.1.7 RGBInit

Initializes the Timer and GPIO functionality associated with the RGB LED

Prototype:

```
void
RGBInit(uint32_t ui32Enable)
```

Parameters:
ui32Enable enables RGB immediately if set.

Description:
This function must be called during application initialization to configure the GPIO pins to which the LEDs are attached. It enables the port used by the LEDs and configures each color's Timer. It optionally enables the RGB LED by configuring the GPIO pins and starting the timers.

Returns:
None.

4.2.1.8 RGBIntensitySet

Set the current output intensity.

Prototype:
```
void
RGBIntensitySet(float fIntensity)
```

Parameters:
fIntensity is used to scale the intensity of all three colors by the same amount. fIntensity should be between 0.0 and 1.0. This scale factor is applied individually to all three colors.

Description:
This function should be called by the application to set the intensity of the RGB LED.

Returns:
None.

4.2.1.9 RGBSet

Set the output color and intensity.

Prototype:
```
void
RGBSet(volatile uint32_t *pui32RGBColor,
       float fIntensity)
```

Parameters:
pui32RGBColor points to a three element array representing the relative intensity of each color. Red is element 0, Green is element 1, Blue is element 2. 0x0000 is off. 0xFFFF is fully on.
fIntensity is used to scale the intensity of all three colors by the same amount. fIntensity should be between 0.0 and 1.0. This scale factor is applied to all three colors.

Description:
This function should be called by the application to set the color and intensity of the RGB LED.

Returns:
None.

4.3 Programming Example

The following example shows how to use the rgb driver to initialize the RGB LED.

```
unsigned long ulColors[3];

//
// Initialize the rgb driver.
//
RGBInit(0);

//
// Set the intensity level from 0.0f to 1.0f
//
    RGBIntensitySet(0.3f);

//
// Initialize the three color values.
//
ulColors[BLUE]  = 0x00FF;
ulColors[RED]   = 0xFFFF;
ulColors[GREEN] = 0x0000;
RGBColorSet(ulColors);

//
// Enable the RGB.  This configure GPIOs to the Timer PWM mode needed
// to generate the color spectrum.
//
RGBEnable();


//
// Application may now call RGB API to suit program requirements.
//
...
```

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