DK-TM4C129X-BOOST-CC3000 Firmware Development Package

USER'S GUIDE

SW-DK-TM4C129X-BOOST-CC3000-UG-2.1.1.71

Copyright

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

Revision Information

This is version 2.1.1.71 of this document, last updated on May 07, 2015.

Table of Contents

Copyright **2**
Revision Information **2**
1 Introduction **5**
2 Example Applications **7**
2.1 CC3000 Basic WiFi Example (cc3000_basic_wifi_application) 7
2.2 CC3000 Firmware Patch Programmer (cc3000_patch_programmer) 8
2.3 CC3000 WiFi Access Point SSID Scanning Example (cc3000_ssid_scan) 9
3 Frame Module **11**
3.1 Introduction 11
3.2 API Functions 11
3.3 Programming Example 12
4 Kentec 320x240x16 Display Driver **13**
4.1 Introduction 13
4.2 API Functions 13
4.3 Programming Example 14
5 MX66L51235F Driver **17**
5.1 Introduction 17
5.2 API Functions 17
5.3 Programming Example 17
6 Pinout Module **19**
6.1 Introduction 19
6.2 API Functions 19
6.3 Programming Example 19
7 Sound Driver **21**
7.1 Introduction 21
7.2 API Functions 21
7.3 Programming Example 21
8 Touch Screen Driver **23**
8.1 Introduction 23
8.2 API Functions 24
8.3 Programming Example 25
IMPORTANT NOTICE **28**

1 Introduction

The Texas Instruments® Tiva™ DK-TM4C129X-BOOST-CC3000 development 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 Tiva microcontroller.

The DK-TM4C129X-BOOST-CC3000 includes a Tiva ARM® Cortex™-M4-based microcontroller and the following features:

- Tiva™ TM4C129XNCZAD microcontroller
- TFT display (320x240 16 bpp) with capacitive touch screen overlay
- Ethernet connector
- USB OTG connector
- 64 MB SPI flash
- MicroSD card connector
- Temperature sensor
- Speaker with class A/B amplifier
- 3 user buttons
- User LED
- 2 booster pack connectors
- EM connector
- On-board In-Circuit Debug Interface (ICDI)
- Power supply option from USB ICDI connection or external power connection
- Shunt for microcontroller current consumption measurement

This document describes the board-specific drivers and example applications that are provided for this development board when paired with the BOOST-CC3000 BoosterPack.

2 Example Applications

The example applications show how to utilize features of the DK-TM4C129X development board. Examples are included to show how to use many of the general features of the Tiva microcontroller, as well as the feature that are unique to this development board.

A number of drivers are provided to make it easier to use the features of the DK-TM4C129X. These drivers also contain low-level code that make use of the TivaWare peripheral driver library and utilities.

There is an IAR workspace file (`dk-tm4c129x-boost-cc3000.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 (`dk-tm4c129x-boost-cc3000.mpw`) 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/dk-tm4c129x-boost-cc3000` subdirectory of the firmware development package source distribution.

2.1 CC3000 Basic WiFi Example (cc3000_basic_wifi_application)

This is a basic WiFi application for the CC3000 BoosterPack. This application is a command line wrapper for various functions that the CC3000 can provide. Please refer to the CC3000 wiki at `http://processors.wiki.ti.com/index.php/CC3000` for more information on the commands provided.

To see available commands type “help” at the serial terminal prompt. The terminal is connected in 8-N-1 mode at 115200 baud.

This example defaults to using BoosterPack1. If you would like to use BoosterPack2 instead please change the define CC3000_USE_BOOSTERPACK1 in the project settings to CC3000_USE_BOOSTERPACK2 and rebuild. When using the BoosterPack2 connector, make sure that jumpers J16 and J17 are both set to the SPI position.

To use this example you must first connect to an existing unencrypted wireless network. This can be done by using the “smartconfig” command with the associated smartphone application. Alternatively, the connection can be made manually by using the ’connect’ command. Once connected you can do any of the following.

Configure an IP address:

1. To use DHCP to allocate a dynamic IP address “ipconfig” or “ipconfig 0 0 0” or,
2. To allocate a static IP address use “ipconfig a.b.c.d” where “a.b.c.d” is the required, dotted-decimal format address.

Send and receive UDP data:

1. Open a UDP socket “socketopen UDP”.
2. Bind the socket to a local port “bind 8080”.
3. Send or receive data “senddata 192.168.1.101 8080 helloworld” or “receivedata”. In the senddata case, the provided parameters identify the IP address of the remote host and the remote

port number to which the data is to be sent.

Send and receive TCP data:

1. Open a TCP socket “socketopen TCP”.
2. Bind the socket to a local port “bind 8080”.
3. Send a request to the remote server “senddata 192.168.1.101 8080 helloworld”. On the first “senddata” after opening the socket, the socket is connected to the specified remote host and port. On further “senddata” requests, the remote address and port are ignored and the existing connection is used.
4. Receive data from the remote server “receivedata”.

Note that, in the current implementation, the application only supports acting as a TCP client. The CC3000 also supports incoming connections as required to operate as a TCP server but this example does not yet include support for this feature.

Send mDNS advertisement:

1. “mdnsadvertise cc3000”

Close the open socket:

1. “socketclose”

Disconnect from network:

1. “disconnect”

Reset the CC3000:

1. “resetcc3000”

Delete connection policy:

This deletes the connection policy from CC3000 memory so that the device won’t auto connect whenever it is reset in future.

1. “deletepolicy”

2.2 CC3000 Firmware Patch Programmer (cc3000_patch_programmer)

This is the Patch Programmer tool for the CC3000 BoosterPack running on an DK-TM4C129X Development Kit. Run the application to download new firmware and driver patches to the CC3000 processor. Status is output on the LCD display and also via UART0 which is available via the virtual COM port provided by the ICDI debug interface.

Two patches are downloaded using this tool with the patch data is linked directly into the application binary. The driver patch can be found in an array named “wlan_drv_patch” and the firmware patch can be found in “fw_patch”. When new patches are available, these arrays must be replaced with

versions containing those new patches and then the application rebuilt and run to apply the patches to the CC3000 hardware.

To view UART0 output from the application, set your host system's serial terminal to use 115200bps, 8-N-1.

By default, the application is built to support a CC3000 BoosterPack connected to the BoosterPack1 connector of the DK-TM4C129X board.

The application may be rebuilt to support a CC3000 BoosterPack connected to the board's BoosterPack 1 or BoosterPack 2 connector, or for a CC3000 Evaluation Module (EM) connected to the board's EM connectors by setting one of the labels "CC3000_USE_BOOSTERPACK1", "CC3000_USE_BOOSTERPACK2" or "CC3000_USE_EM" in the build environment. When connecting the CC3000 to BoosterPack2, jumpers J16 and J17 must be moved to ensure that SPI signals (rather than I2C) are routed to the appropriate BoosterPack connector pins.

For information on the CC3000 software stack and API, please consult the wiki at `http://processors.wiki.ti.com/index.php/CC3000`.

2.3 CC3000 WiFi Access Point SSID Scanning Example (cc3000_ssid_scan)

This example requires a CC3000 WiFi BoosterPack attached to the the DK-TM4C129X Development Kit. After booting and initializing the CC3000, the application initiates a WiFi scan for access points. When the scan completes, the SSID, BSSID and security protocol supported by each detected access point are shown. Another scan can be started by pressing the "Scan" button on the touchscreen display.

By default, the application is built to support a CC3000 BoosterPack connected to the BoosterPack1 connector of the DK-TM4C129X board.

The application may be rebuilt to support a CC3000 BoosterPack connected to the board's BoosterPack 1 or BoosterPack 2 connector or a CC3000 EM connected to the board's EM connectors by setting one of the labels "CC3000_USE_BOOSTERPACK1", "CC3000_USE_BOOSTERPACK2" or "CC3000_USE_EM" in the build environment. When connecting the CC3000 to BoosterPack2, jumpers J16 and J17 must be moved to ensure that SPI signals (rather than I2C) are routed to the appropriate BoosterPack connector pins.

For information on the CC3000 software stack and API, please consult the wiki at `http://processors.wiki.ti.com/index.php/CC3000`.

3 Frame Module

Introduction 11
API Functions 11
Programming Example 12

3.1 Introduction

The frame module is a common function for drawing an application frame on the display. This is used by the example applications to provide a uniform appearance.

This driver is located in `examples/boards/dk-tm4c129x-boost-cc3000/drivers`, with `frame.c` containing the source code and `frame.h` containing the API declarations for use by applications.

3.2 API Functions

Functions

- void FrameDraw (tContext *psContext, const char *pcAppName)

3.2.1 Function Documentation

3.2.1.1 FrameDraw

Draws a frame on the LCD with the application name in the title bar.

Prototype:

```
void
FrameDraw(tContext *psContext,
          const char *pcAppName)
```

Parameters:

psContext is a pointer to the graphics library context used to draw the application frame.

pcAppName is a pointer to a string that contains the name of the application.

Description:

This function draws an application frame onto the LCD, using the supplied graphics library context to access the LCD and the given name in the title bar of the application frame.

Returns:

None.

3.3 Programming Example

The following example shows how to draw the application frame.

```
//
// The frame example.
//
void
FrameExample(void)
{
    tContext sContext;

    //
    // Draw the application frame.  This code assumes the the graphics library
    // context has already been initialized.
    //
    FrameDraw(&sContext, "example");
}
```

4 Kentec 320x240x16 Display Driver

Introduction 13
API Functions 13
Programming Example 14

4.1 Introduction

The display driver offers a standard interface to access display functions on the Kentec K350QVG-V2-F 320x240 16-bit color TFT display and is used by the TivaWare Graphics Library and widget manager. The display is controlled by the embedded SSD2119 display controller, which provides the frame buffer for the display. In addition to providing the `tDisplay` structure required by the graphics library, the display driver also provides an API for initializing the display.

The display driver can be built to operate in one of four orientations:

- **LANDSCAPE** - In this orientation, the screen is wider than it is tall; this is the normal orientation for a television or a computer monitor, and is the normal orientation for photographs of the outdoors (hence the name). For the K350QVG-V2-F, the flex connector is on the bottom side of the screen when viewed in **LANDSCAPE** orientation.
- **PORTRAIT** - In this orientation, the screen is taller than it is wide; this is the normal orientation of photographs of people (hence the name). For the K350QVG-V2-F, the flex connector is on the left side of the screen when viewed in **PORTRAIT** orientation.
- **LANDSCAPE_FLIP** - **LANDSCAPE** mode rotated 180 degrees (in other words, the flex connector is on the top side of the screen).
- **PORTRAIT_FLIP** - **PORTRAIT** mode rotated 180 degrees (in other words, the flex connector is on the right side of the screen).

One of the above highlighed defines selects the orientation that the display driver will use. If none is defined, the default orientation is **LANDSCAPE_FLIP** (which corresponds to how the display is mounted to the DK-TM4C129X development board).

This driver is located in `examples/boards/dk-tm4c129x-boost-cc3000/drivers`, with `kentec320x240x16_ssd2119.c` containing the source code and `kentec320x240x16_ssd2119.h` containing the API declarations for use by applications.

4.2 API Functions

Functions

- void Kentec320x240x16_SSD2119Init (uint32_t ui32SysClock)

Variables

- const tDisplay g_sKentec320x240x16_SSD2119

4.2.1 Function Documentation

4.2.1.1 Kentec320x240x16_SSD2119Init

Initializes the display driver.

Prototype:

```
void
Kentec320x240x16_SSD2119Init(uint32_t ui32SysClock)
```

Parameters:

ui32SysClock is the frequency of the system clock.

Description:

This function initializes the LCD controller and the SSD2119 display controller on the panel, preparing it to display data.

Returns:

None.

4.2.2 Variable Documentation

4.2.2.1 g_sKentec320x240x16_SSD2119

Definition:

```
const tDisplay g_sKentec320x240x16_SSD2119
```

Description:

The display structure that describes the driver for the Kentec K350QVG-V2-F TFT panel with an SSD2119 controller.

4.3 Programming Example

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

```
//
// The Kentec 320x240x16 SSD2119 example.
//
void
Kentec320x240x16_SSD2119Example(void)
{
    uint32_t ui32SysClock;
    tContext sContext;
```

```
    //
    // Initialize the display.  This code assumes that ui32SysClock has been
    // set to the clock frequency of the device (for example, the value
    // returned by SysCtlClockFreqSet).
    //
    Kentec320x240x16_SSD2119Init(ui32SysClock);

    //
    // Initialize a graphics library drawing context.
    //
    GrContextInit(&sContext, &g_sKentec320x240x16_SSD2119);
}
```

5 MX66L51235F Driver

Introduction 17
API Functions 17
Programming Example 17

5.1 Introduction

The MX66l51235F driver provides functions to make it easy to use the MX66L51235F SPI flash on the DK-TM4C129X development board. The driver provides a function to read, erase, and program the SPI flash.

On the DK-TM4C129X development board, the SPI flash shares a SPI port with the SD card socket. If not properly initialized into SPI mode, the SD card will occasionally drive data onto the SPI bus despite the fact that it is “not selected” (which is in fact valid since there is not chip select for an SD card in SD card mode). Therefore, the SD card must be properly initialized (via a call to the disk_initialize() function in the SD card driver) prior to using this driver to access the SPI flash.

This driver is located in `examples/boards/dk-tm4c129x-boost-cc3000/drivers`, with `mx66l51235f.c` containing the source code and `mx66l51235f.h` containing the API declarations for use by applications.

5.2 API Functions

5.3 Programming Example

The following example shows how to use the MX66L51235F driver to read and write data in the SPI flash.

```
//
// A buffer to hold the data read from and written to the SPI flash.
//
uint8_t g_pui8MX66L51235FData[32];

//
// The MX66L51235F example.
//
void
MX66L51235FExample(void)
{
    uint32_t ui32SysClock;

    //
    // Initialize the SPI flash driver.  This code assumes that ui32SysClock
    // has been set to the clock frequency of the device (for example, the
    // value returned by SysCtlClockFreqSet).
    //
    MX66L51235FInit(ui32SysClock);

    //
    // Erase the first sector (4 KB) of the SPI flash.
```

```
    //
    MX66L51235FSectorErase(0);

    //
    // Program some data into the first page of the SPI flash.  This assumes
    // that the data buffer has been filled with the data to be programmed.
    //
    MX66L51235FPageProgram(0, g_pui8MX66L51235FData,
                           sizeof(g_pui8MX66L51235FData));

    //
    // Read some data from the second page of the SPI flash.
    //
    MX66L51235FRead(0x100, g_pui8MX66L51235FData,
                    sizeof(g_pui8MX66L51235FData));
}
```

6 Pinout Module

Introduction 19
API Functions 19
Programming Example 19

6.1 Introduction

The pinout module is a common function for configuring the device pins for use by example applications. The pins are configured into the most common usage; it is possible that some of the pins might need to be reconfigured in order to support more specialized usage.

This driver is located in `examples/boards/dk-tm4c129x-boost-cc3000/drivers`, with `pinout.c` containing the source code and `pinout.h` containing the API declarations for use by applications.

6.2 API Functions

Functions

- void PinoutSet (void)

6.2.1 Function Documentation

6.2.1.1 PinoutSet

Configures the device pins for the standard usages on the DK-TM4C129X.

Prototype:

```
void
PinoutSet(void)
```

Description:

This function enables the GPIO modules and configures the device pins for the default, standard usages on the DK-TM4C129X. Applications that require alternate configurations of the device pins can either not call this function and take full responsibility for configuring all the device pins, or can reconfigure the required device pins after calling this function.

Returns:

None.

6.3 Programming Example

The following example shows how to configure the device pins.

```
//
// The pinout example.
//
void
PinoutExample(void)
{
    //
    // Configure the device pins.
    //
    PinoutSet();
}
```

7 Sound Driver

Introduction 21
API Functions 21
Programming Example 21

7.1 Introduction

The sound driver provides a set of functions to stream 16-bit PCM audio data to the speaker on the DK-TM4C129X development board. The audio can be played at 8 kHz, 16 kHz, 32 kHz, or 64 kHz; in each case the data is output to the speaker at 64 kHz, and at lower playback rates the intervening samples are computed via linear interpolation (which is fast but introduces high-frequency artifacts).

The audio data is supplied via a ping-pong buffer. This is a buffer that is logically split into two halves; the “ping” half and the “pong” half. While the sound driver is playing data from one half, the application is responsible for filling the other half with new audio data. A callback from the sound driver indicates when it transitions from one half to the other, which provides the indication that one of the halves has been consumed and must be filled with new data.

The sound driver utilizes timer 5 subtimer A. The interrupt from the timer 5 subtimer A is used to process the audio stream; the SoundIntHandler() function should be called when this interrupt occurs (which is typically accomplished by placing it in the vector table in the startup code for the application).

This driver is located in `examples/boards/dk-tm4c129x-boost-cc3000/drivers`, with `sound.c` containing the source code and `sound.h` containing the API declarations for use by applications.

7.2 API Functions

7.3 Programming Example

The following example shows how to use the sound driver to playback a stream of 8 kHz audio data.

```
//
// The buffer used as the audio ping-pong buffer.
//
#define SOUND_NUM_SAMPLES       256
int16_t g_pi16SoundExampleBuffer[SOUND_NUM_SAMPLES];

//
// The callback function for when half of the buffer has been consumed.
//
void
SoundExampleCallback(uint32_t ui32Half)
{
    //
    // Generate new audio in the half the has just been consumed.  As this is
    // in the context of the timer interrupt, it is possible/likely that this
```

```
    // should just set a flag to trigger something else (outside of interrupt
    // context) to do the actual work.  In either case, this needs to return
    // prior to the next timer interrupt (in other words, within ~15 us).
    //
}

//
// The sound example.
//
void
SoundExample(void)
{
    uint32_t ui32SysClock;

    //
    // Initialize the sound driver.  This code assumes that ui32SysClock has
    // been set to the clock frequency of the device (for example, the value
    // returned by SysCtlClockFreqSet).
    //
    SoundInit(ui32SysClock);

    //
    // Prefill the audio buffer with the first segment of the audio to be
    // played.
    //

    //
    // Start the playback of audio.
    //
    SoundStart(g_pi16SoundExampleBuffer, SOUND_NUM_SAMPLES, 8000,
               SoundExampleCallback);
}
```

8 Touch Screen Driver

Introduction 23
API Functions 24
Programming Example 25

8.1 Introduction

The touch screen is a pair of resistive layers on the surface of the display. One layer has connection points at the top and bottom of the screen, and the other layer has connection points at the left and right of the screen. When the screen is touched, the two layers make contact and electricity can flow between them.

The horizontal position of a touch can be found by applying positive voltage to the right side of the horizontal layer and negative voltage to to the left side. When not driving the top and bottom of the vertical layer, the voltage potential on that layer will be proportional to the horizontal distance across the screen of the press, which can be measured with an ADC channel. By reversing these connections, the vertical position can also be measured. When the screen is not being touched, there will be no voltage on the non-powered layer.

By monitoring the voltage on each layer when the other layer is appropriately driven, touches and releases on the screen, as well as movements of the touch, can be detected and reported.

In order to read the current voltage on the two layers and also drive the appropriate voltages onto the layers, each side of each layer is connected to both a GPIO and an ADC channel. The GPIO is used to drive the node to a particular voltage, and when the GPIO is configured as an input, the corresponding ADC channel can be used to read the layer's voltage.

The touch screen is sampled every 2.5 ms, with four samples required to properly read both the X and Y position. Therefore, 100 X/Y sample pairs are captured every second.

Like the display driver, the touch screen driver operates in the same four orientations (selected in the same manner). Default calibrations are provided for using the touch screen in each orientation; the calibrate application can be used to determine new calibration values if necessary.

The touch screen driver utilizes sample sequence 3 of ADC0 and timer 5 subtimer B. The interrupt from the ADC0 sample sequence 3 is used to process the touch screen readings; the TouchScreenIntHandler() function should be called when this interrupt occurs (which is typically accomplished by placing it in the vector table in the startup code for the application).

The touch screen driver makes use of calibration parameters determined using the "calibrate" example application. The theory behind these parameters is explained by Carlos E. Videles in the June 2002 issue of Embedded Systems Design. It can be found online at `http://www.embedded.com/story/OEG20020529S0046`.

This driver is located in `examples/boards/dk-tm4c129x-boost-cc3000/drivers`, with `touch.c` containing the source code and `touch.h` containing the API declarations for use by applications.

8.2 API Functions

Functions

- void TouchScreenCallbackSet (int32_t (*pfnCallback)(uint32_t ui32Message, int32_t i32X, int32_t i32Y))
- void TouchScreenInit (uint32_t ui32SysClock)
- void TouchScreenIntHandler (void)

8.2.1 Function Documentation

8.2.1.1 TouchScreenCallbackSet

Sets the callback function for touch screen events.

Prototype:

```
void
TouchScreenCallbackSet(int32_t (*ui32Message,)(uint32_t int32_t i32X,
int32_t i32Y) pfnCallback)
```

Parameters:

pfnCallback is a pointer to the function to be called when touch screen events occur.

Description:

This function sets the address of the function to be called when touch screen events occur. The events that are recognized are the screen being touched (“pen down”), the touch position moving while the screen is touched (“pen move”), and the screen no longer being touched (“pen up”).

Returns:

None.

8.2.1.2 TouchScreenInit

Initializes the touch screen driver.

Prototype:

```
void
TouchScreenInit(uint32_t ui32SysClock)
```

Parameters:

ui32SysClock is the frequency of the system clock.

Description:

This function initializes the touch screen driver, beginning the process of reading from the touch screen. This driver uses the following hardware resources:

- ADC 0 sample sequence 3
- Timer 5 subtimer B

Returns:
None.

8.2.1.3 TouchScreenIntHandler

Handles the ADC interrupt for the touch screen.

Prototype:
```
void
TouchScreenIntHandler(void)
```

Description:
This function is called when the ADC sequence that samples the touch screen has completed its acquisition. The touch screen state machine is advanced and the acquired ADC sample is processed appropriately.

It is the responsibility of the application using the touch screen driver to ensure that this function is installed in the interrupt vector table for the ADC0 samples sequencer 3 interrupt.

Returns:
None.

8.3 Programming Example

The following example shows how to initialize the touchscreen driver and the callback function which receives notifications of touch and release events in cases where the StellarisWare Graphics Library widget manager is not being used by the application.

```
//
// The touch screen driver calls this function to report all state changes.
//
static long
TouchTestCallback(uint32_t ui32Message, int32_t i32X, int32_t i32Y)
{
    //
    // Check the message to determine what to do.
    //
    switch(ui32Message)
    {
        //
        // The screen is no longer being touched (in other words, pen/pointer
        // up).
        //
        case WIDGET_MSG_PTR_UP:
        {
            //
            // Handle the pointer up message if required.
            //
            break;
        }

        //
        // The screen has just been touched (in other words, pen/pointer down).
        //
        case WIDGET_MSG_PTR_DOWN:
```

```
            {
                //
                // Handle the pointer down message if required.
                //
                break;
            }

            //
            // The location of the touch on the screen has moved (in other words,
            // the pen/pointer has moved).
            //
            case WIDGET_MSG_PTR_MOVE:
            {
                //
                // Handle the pointer move message if required.
                //
                break;
            }

            //
            // An unknown message was received.
            //
            default:
            {
                //
                // Ignore all unknown messages.
                //
                break;
            }
        }

        //
        // Success.
        //
        return(0);
    }

    //
    // The first touch screen example.
    //
    void
    TouchScreenExample1(void)
    {
        uint32_t ui32SysClock;

        //
        // Initialize the touch screen driver.  This code assumes that ui32SysClock
        // has been set to the clock frequency of the device (for example, the
        // value returned by SysCtlClockFreqSet).
        //
        TouchScreenInit(ui32SysClock);

        //
        // Register the application callback function that is to receive touch
        // screen messages.
        //
        TouchScreenCallbackSet(TouchTestCallback);
    }
```

If using the StellarisWare Graphics Library widget manager, touchscreen initialization code is as follows. In this case, the touchscreen callback is provided within the widget manager so no additional function is required in the application code.

```
    //
```

```
// The second touch screen example.
//
void
TouchScreenExample2(void)
{
    uint32_t ui32SysClock;

    //
    // Initialize the touch screen driver.  This code assumes that ui32SysClock
    // has been set to the clock frequency of the device (for example, the
    // value returned by SysCtlClockFreqSet).
    //
    TouchScreenInit(ui32SysClock);

    //
    // Register the graphics library pointer message callback function so that
    // it receives touch screen messages.
    //
    TouchScreenCallbackSet(WidgetPointerMessage);
}
```

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.

"So long and thanks for all the fish." - Douglas Adams

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