EK-TM4C123GXL-BOOSTXL-SENSHUB Firmware Development Package

USER'S GUIDE

SW-EK-TM4C123GXL-BOOSTXL-SENSHUB-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 is a low cost platform that can be used for software development and to prototype a hardware design. A variety of BoosterPacks are available to quickly extend the LaunchPads features.

This document describes the example applications that are provided for the EK-TM4C123GXL when paired with the BOOSTXL-SENSHUB BoosterPack. This BoosterPack provides a variety of motion and environmental sensors. It also provides an EM expansion option for attachement of additional peripherals such as the CC2533EMK or CC4000EMK. These examples utilize the TivaWare™ for C Series Sensor Library to extract and process information from the BOOSTXL-SENSHUB.

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-boostxl-senshub.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-boostxl-senshub.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-boostxl-senshub` subdirectory of the firmware development package source distribution.

2.1 Motion Air Mouse (airmouse)

This example demonstrates the use of the Sensor Library, TM4C123G LaunchPad and the SensHub BoosterPack to fuse nine axis sensor measurements into motion and gesture events. These events are then transformed into mouse and keyboard events to perform standard HID tasks.

Connect the device USB port on the side of the LaunchPad to a standard computer USB port. The LaunchPad with SensHub BoosterPack enumerates on the USB bus as a composite HID keyboard and mouse.

Hold the LaunchPad with the buttons away from the user and toward the computer with USB Device cable exiting the right and bottom corner of the board.

- Roll or tilt the LaunchPad to move the mouse cursor of the computer up, down, left and right.
- The buttons on the LaunchPad perform the left and right mouse click actions. The buttons on the SensHub BoosterPack are not currently used by this example.
- A quick spin of the LaunchPad generates a PAGE_UP or PAGE_DOWN keyboard press and release depending on the direction of the spin. This motion simulates scrolling.
- A quick horizontal jerk to the left or right generates a CTRL+ or CTRL- keyboard event, which creates the zoom effect used in many applications, especially web browsers.
- A quick vertical lift generates an ALT+TAB keyboard event, which allows the computer user to select between currently open windows.
- A quick twist to the left or right moves the window selector.
- A quick jerk in the down direction selects the desired window and closes the window selection dialog.

This example also supports the RemoTI low power RF Zigbee® human interface device profile. The wireless features of this example require the CC2533EMK expansion card and the CC2531EMK USB Dongle. For details and instructions for wireless operations see the

Wiki at http://processors.wiki.ti.com/index.php/Tiva_C_Series_LaunchPad and http://processors.wiki.ti.com/index.php/Wireless_Air_Mouse_Guide.

2.2 Nine Axis Sensor Fusion with the MPU9150 and Complimentary-Filtered DCM (compdcm_mpu9150)

This example demonstrates the basic use of the Sensor Library, TM4C123G LaunchPad and SensHub BoosterPack to obtain nine axis motion measurements from the MPU9150. The example fuses the nine axis measurements into a set of Euler angles: roll, pitch and yaw. It also produces the rotation quaternions. The fusion mechanism demonstrated is complimentary-filtered direct cosine matrix (DCM) algorithm is provided as part of the Sensor Library.

Connect a serial terminal program to the LaunchPad's ICDI virtual serial port at 115,200 baud. Use eight bits per byte, no parity and one stop bit. The raw sensor measurements, Euler angles and quaternions are printed to the terminal. The RGB LED begins to blink at 1Hz after initialization is completed and the example application is running.

2.3 Humidity Measurement with the SHT21 (humidity_sht21)

This example demonstrates the basic use of the Sensoror Library, TM4C123G LaunchPad and SensHub BoosterPack to obtain temperature and relative humidity of the environment using the Sensirion SHT21 sensor.

Connect a serial terminal program to the LaunchPad's ICDI virtual serial port at 115,200 baud. Use eight bits per byte, no parity and one stop bit. The humidity and temperature as measured by the SHT21 is printed to the terminal. The RGB LED begins to blink at 1Hz after initialization is complete and the example application is running.

2.4 Humidity Measurement with the SHT21 (humidity_sht21_simple)

This example demonstrates the usage of the I2C interface to obtain the temperature and relative humidity of the environment using the Sensirion SHT21 sensor.

The I2C3 on the EK-TM4C123GXL launchPad is used to interface with the SHT21 sensor. The SHT21 sensor is on the BOOSTXL_SENSHUB boosterPack expansion board that can be directly plugged into the Booster Pack connector on the EK-TM4C123GXL launchPad board.

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

- I2C3 peripheral
- GPIO Port D peripheral
- I2C3_SCL - PD0

- I2C3_SDA - PD1

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

2.5 Light Measurement with the ISL29023 (light_isl29023)

This example demonstrates the basic use of the Sensor Library, TM4C123G LaunchPad and the SensHub BoosterPack to obtain ambient and infrared light measurements with the ISL29023 sensor.

Connect a serial terminal program to the LaunchPad's ICDI virtual serial port at 115,200 baud. Use eight bits per byte, no parity and one stop bit. The raw sensor measurements are printed to the terminal. The RGB LED blinks at 1Hz once the initialization is complete and the example is running.

2.6 Pressure Measurement with the BMP180 (pressure_bmp180)

This example demonstrates the basic use of the Sensor Library, TM4C123G LaunchPad and the SensHub BoosterPack to obtain air pressure and temperature measurements with the BMP180 sensor.

Connect a serial terminal program to the LaunchPad's ICDI virtual serial port at 115,200 baud. Use eight bits per byte, no parity and one stop bit. The raw sensor measurements are printed to the terminal. The RGB LED blinks at 1Hz once the initialization is complete and the example is running.

2.7 Temperature Measurement with the TMP006 (temperature_tmp006)

This example demonstrates the basic use of the Sensor Library, TM4C123G LaunchPad and the SensHub BoosterPack to obtain ambient and object temperature measurements with the Texas Instruments TMP006 sensor.

Connect a serial terminal program to the LaunchPad's ICDI virtual serial port at 115,200 baud. Use eight bits per byte, no parity and one stop bit. The raw sensor measurements are printed to the terminal. The RGB LED blinks at 1Hz once the initialization is complete and the example is running.

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-boostxl-senshub/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-boostxl-senshub/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