EK-TM4C123GXL Firmware Development Package

USER'S GUIDE

SW-EK-TM4C123GXL-UG-2.0

Copyright

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

Revision Information

This is version 2.0 of this document, last updated on August 29, 2013.

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 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).

2.2 Blinky (blinky)

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

2.3 FreeRTOS Example (freertos_demo)

This application demonstrates the use of FreeRTOS on Launchpad.

The application blinks the user-selected LED at a user-selected frequency. To select the LED press the left button and to select the frequency press the right button. The UART outputs the application status at 115,200 baud, 8-n-1 mode.

This application utilizes FreeRTOS to perform the tasks in a concurrent fashion. The following tasks are created:

- An LED task, which blinks the user-selected on-board LED at a user-selected rate (changed via the buttons).
- A Switch task, which monitors the buttons pressed and passes the information to LED task.

In addition to the tasks, this application also uses the following FreeRTOS resources:

- A Queue to enable information transfer between tasks.

- A Semaphore to guard the resource, UART, from access by multiple tasks at the same time.
- A non-blocking FreeRTOS Delay to put the tasks in blocked state when they have nothing to do.

For additional details on FreeRTOS, refer to the FreeRTOS web page at: `http://www.freertos.org/`

2.4 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.

2.5 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.

2.6 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).

2.7 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.

2.8 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. See `http://www.ti.com/tm4c123g-launchpad/project0` for more information and tutorial videos.

2.9 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.

2.10 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.

2.11 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.

2.12 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.

2.13 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-for-C-Series/windows_drivers directory of TivaWare C series releases. This INF contains information required to install the WinUSB subsystem on Windowi16XP 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 ”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.

2.14 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 C Series 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-for-C-Series/windows_drivers. For Windows 2000, the required INF file is in C:/ti/TivaWare-for-C-Series/windows_drivers/win2K.

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 buttons driver to initialize the buttons, debounce and read the buttons state.

```
unsigned long ulColors[3];

//
// Initialize the buttons.
//
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.
//
...
```