EK-TM4C1294XL Firmware Development Package

USER'S GUIDE

SW-EK-TM4C1294XL-UG-2.2.0.295

Copyright © 2013-2020
Texas Instruments Incorporated

Copyright

Copyright © 2013-2020 Texas Instruments Incorporated. All rights reserved. Tiva, TivaWare, Code Composer Studio are trademarks of Texas Instruments. Arm, Cortex, Thumb are registered trademarks of Arm Limited. All other trademarks are the property of their respective owners.

Please be aware that an important notice concerning availability, standard warranty, and use in critical applications of Texas Instruments semiconductor products and disclaimers thereto appears at the end of this document.

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™ EK-TM4C1294XL evaluation board (Tiva C Series TM4C1294 Connected LaunchPad) is a low cost platform that can be used for software development and prototyping a hardware design. A variety of BoosterPacks are available to quickly extend the LaunchPad's features.

The EK-TM4C1294XL includes a Tiva ARM® Cortex™-M4-based microcontroller and the following features:

- Tiva™ TM4C1294NCPDT microcontroller
- Ethernet connector
- USB OTG connector
- 2 user buttons
- 4 User LEDs
- 2 BoosterPack XL sites
- On-board In-Circuit Debug Interface (ICDI)
- Power supply option from USB ICDI connection, USB OTG connection or external power connection
- Shunt jumper for microcontroller current consumption measurement

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

2 Example Applications

The example applications show how to utilize features of this evaluation 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 evaluation board.

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

There is an IAR workspace file (`ek-tm4c1294xl.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

There is a Keil multi-project workspace file (`ek-tm4c1294xl.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/ek-tm4c1294xl` subdirectory of the firmware development package source distribution.

2.1 ADC with uDMA Demo (adc_udma)

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 - PE3

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 peripheral, wait for SW1 to be pressed and blink LED D1. When the SW1 is pressed, LED D1 is turned off, and then the application branches to the 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.

This application is intended for use with the boot_serial flash-based boot loader included in the software release. Since the sector size is 16KB, the link address is set to 0x4000. If you are using USB or Ethernet boot loader, you may change this address to a 16KB 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 TM4C129x-class device also support serial, Ethernet and USB boot loaders in ROM. To make use of this function, link your application to run at address 0x0000 in flash and enter the bootloader using either the ROM_UpdateSerial, ROM_UpdateEMAC or ROM_UpdateUSB functions (defined in rom.h). This mechanism is used in the utils/swupdate.c module when built specifically targeting a suitable TM4C129x-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 peripheral, wait for SW1 to be pressed and blink LED D2. When the SW1 is pressed, LED D2 is turned off, and then the application branches to the 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.

This application is intended for use with the boot_serial flash-based boot loader included in the software release. Since the sector size is 16KB, the link address is set to 0x4000. If you are using USB or Ethernet boot loader, you may change this address to a 16KB 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 TM4C129x-class device also support serial, Ethernet and USB boot loaders in ROM. To make use of this function, link your application to run at address 0x0000 in flash and enter the bootloader using either the ROM_UpdateSerial, ROM_UpdateEMAC or ROM_UpdateUSB functions (defined in rom.h). This mechanism is used in the utils/swupdate.c module when built specifically targeting a suitable TM4C129x-class device.

2.6 Boot Loader ethernet Demo (boot_demo_emac_flash)

An example to demonstrate the use of remote update signaling with the flash-based Ethernet boot loader. This application configures the Ethernet controller and acquires an IP address which is displayed on the screen along with the board's MAC address. It then listens for a "magic packet" telling it that a firmware upgrade request is being made and, when this packet is received, transfers control into the boot loader to perform the upgrade.

This application is intended for use with flash-based ethernet boot loader (boot_emac).

The link address for this application is set to 0x4000, the link address has to be multiple of the flash erase block size(16KB=0x4000). You may change this address to a 16KB 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_demo_flash application can be used along with this application to easily demonstrate that the boot loader is actually updating the on-chip flash.

2.7 ROM Boot Loader Ethernet Demo (boot_demo_emac_rom)

An example to demonstrate the use of remote update signaling with the ROM-based Ethernet boot loader. This application configures the Ethernet controller and acquires an IP address. It then listens for a "magic packet" telling it that a firmware upgrade request is being made and, when this packet is received, transfers control to the ROM boot loader to perform the upgrade.

2.8 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.

2.9 Boot Loader USB Demo (boot_demo_usb)

This example application is used in conjunction with the USB boot loader in ROM and turns the development board into a composite device supporting a mouse via the Human Interface Device class and also publishing runtime Device Firmware Upgrade (DFU) capability. Pressing USR_SW1 will trigger the USB mouse interface to move the cursor in a square pattern once. This is a basic example of how HID reports are sent to the USB host in order to demonstrate controlling the mouse pointer on the host system.

Since the device also publishes a DFU interface, host software such as the dfuprog tool can determine that the device is capable of receiving software updates over USB. The runtime DFU protocol allows such tools to signal the device to switch into DFU mode and prepare to receive a new software image.

Runtime DFU functionality requires only that the device listen for a particular request (DETACH) from the host and, when this is received, transfer control to the USB boot loader via the normal means to re-enumerate as a pure DFU device capable of uploading and downloading firmware images.

Windows device drivers for both the runtime and DFU mode of operation 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 the `C:/Program Files/Texas Instruments/Tiva/usb_examples` or `C:/Program Files (x86)/Texas Instruments/Tiva/usb_examples` directory.

With the device connected to your PC and the device driver installed, enter the following command to enumerate DFU devices:

```
dfuprog -e
```

This will list all DFU-capable devices found and you should see that you have one or two devices available which are in “Runtime” mode.

IMPORTANT - PLEASE READ *** If you see two devices, it is strongly recommended that you disconnect ICDI debug port from the PC, and change the POWER_SELECT jumper (JP1) from ’ICDI’ to ’OTG’ in order to power the LaunchPad from the USB OTG port. The reason for this is that the ICDI chip on the board is a DFU-capable TM4C129x device, and if not careful, the firmware on the ICDI chip could be accidently erased which can not restored easily. As a result, debug capabilities would be lost! IMPORTANT - PLEASE READ ***

If IDCI debug port is disconnected from your PC, you should see only one device from above command, and its index should be 0, and should be named as “Mouse with Device Firmware Upgrade”. If for any reason you need to keep the ICDI port connected, the above command should show two devices. The second device is probably named as “In-Circuit Debug interface”, and we need to be careful not to update the firmware on that device. So please take careful note of the index for the device “Mouse with Device Firmware Upgrade”, it could be 0 or 1, we will need this index number for the following command. Entering the following command will switch this device into DFU mode and leave it ready to receive a new firmware image:

```
dfuprog -i index -m
```

After entering this command, you should notice that the device disconnects from the USB bus and reconnects again. Running “`dfuprog -e`” a second time will show that the device is now in DFU mode and ready to receive downloads. At this point, either LM Flash Programmer or dfuprog may be used to send a new application binary to the device.

2.10 Ethernet flash-based Boot Loader (boot_emac_flash)

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

microcontroller, utilizing either UART0, USB, or Ethernet. The capabilities of the boot loader are configured via the bl_config.h include file. For this example, the boot loader uses Ethernet to load an application.

The configuration is set to boot applications which are linked to run from address 0x4000 in flash. This is minimal address since the flash erase size is 16K bytes.

Please note that LMFlash programmer version number needs to be at least 1588 or later. Older LMFlash programmer doesn't work with this Ethernet boot loader.

2.11 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 0x4000 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 TM4C1294 and other TM4C129x-class devices also support serial boot loaders in ROM.

2.12 Ethernet-based I/O Control (enet_io)

This example application demonstrates web-based I/O control using the Tiva Ethernet controller and the lwIP TCP/IP Stack. DHCP is used to obtain an Ethernet address. If DHCP times out without obtaining an address, a static IP address will be chosen using AutoIP. The address that is selected will be shown on the UART allowing you to access the internal web pages served by the application via your normal web browser.

Two different methods of controlling board peripherals via web pages are illustrated via pages labeled "IO Control Demo 1" and "IO Control Demo 2" in the navigation menu on the left of the application's home page. In both cases, the example allows you to toggle the state of the user LED on the board and set the speed of a blinking LED.

"IO Control Demo 1" uses JavaScript running in the web browser to send HTTP requests for particular special URLs. These special URLs are intercepted in the file system support layer (io_fs.c) and used to control the LED and animation LED. Responses generated by the board are returned to the browser and inserted into the page HTML dynamically by more JavaScript code.

"IO Control Demo 2" uses standard HTML forms to pass parameters to CGI (Common Gateway Interface) handlers running on the board. These handlers process the form data and control the animation and LED as requested before sending a response page (in this case, the original form) back to the browser. The application registers the names and handlers for each of its CGIs with the HTTPD server during initialization and the server calls these handlers after parsing URL parameters each time one of the CGI URLs is requested.

Information on the state of the various controls in the second demo is inserted into the served HTML

using SSI (Server Side Include) tags which are parsed by the HTTPD server in the application. As with the CGI handlers, the application registers its list of SSI tags and a handler function with the web server during initialization and this handler is called whenever any registered tag is found in a .shtml, .ssi, .shtm or .xml file being served to the browser.

In addition to LED and animation speed control, the second example also allows a line of text to be sent to the board for output to the UART. This is included to illustrate the decoding of HTTP text strings.

Note that the web server used by this example has been modified from the example shipped with the basic lwIP package. Additions include SSI and CGI support along with the ability to have the server automatically insert the HTTP headers rather than having these built in to the files in the file system image.

Source files for the internal file system image can be found in the “fs” directory. If any of these files are changed, the file system image (io_fsdata.h) should be rebuilt by running the following command from the enet_io directory:

../../../../tools/bin/makefsfile -i fs -o io_fsdata.h -r -h -q

For additional details on lwIP, refer to the lwIP web page at: `http://savannah.nongnu.org/projects/lwip/`

2.13 Ethernet with lwIP (enet_lwip)

This example application demonstrates the operation of the Tiva Ethernet controller using the lwIP TCP/IP Stack. DHCP is used to obtain an Ethernet address. If DHCP times out without obtaining an address, AutoIP will be used to obtain a link-local address. The address that is selected will be shown on the UART.

UART0, connected to the ICDI virtual COM port and running at 115,200, 8-N-1, is used to display messages from this application. Use the following command to re-build the any file system files that change.

../../../../tools/bin/makefsfile -i fs -o enet_fsdata.h -r -h -q

For additional details on lwIP, refer to the lwIP web page at: `http://savannah.nongnu.org/projects/lwip/`

2.14 Ethernet TCP Echo Server (enet_tcp_echo_server)

This example application demonstrates the operation of the TM4C129x Ethernet controller using the lwIP TCP/IP Stack. DHCP is used to obtain an Ethernet address. If DHCP times out without obtaining an address, AutoIP will be used to obtain a link-local address. The address that is selected will be shown on the UART. The application echoes back the data received from the client.

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

For additional details on lwIP, refer to the lwIP web page at: `http://savannah.nongnu.org/projects/lwip/`

2.15 Ethernet with uIP (enet_uip)

This example application demonstrates the operation of the Tiva C Series Ethernet controller using the uIP TCP/IP Stack. DHCP is used to obtain an Ethernet address. A basic web site is served over the Ethernet port. The web site displays a few lines of text, and a counter that increments each time the page is sent.

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

For additional details on uIP, refer to the uIP web page at: `http://www.sics.se/~adam/uip/`

2.16 Ethernet Weather Application (enet_weather)

This example application demonstrates the operation of the Tiva C series evaluation kit as a weather reporting application.

The application supports updating weather information from Open Weather Map weather provider(`http://openweathermap.org/`). The application uses the lwIP stack to obtain an address through DNS, resolve the address of the Open Weather Map site and then build and handle all of the requests necessary to access the weather information. The application can also use a web proxy, allows for a custom city to be added to the list of cities and toggles temperature units from Celsius to Fahrenheit. The application scrolls through the city list at a fixed interval when there is a valid IP address and displays this information over the UART.

The application will print the city name, status, humidity, current temp and the high/low. In addition the application will display what city it is currently updating. Once the app has scrolled through the cities a a defined amount of times, it will attempt to update the information.

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

/ For additional details about Open Weather Map refer to their web page at: `http://openweathermap.org/`

For additional details on lwIP, refer to the lwIP web page at: `http://savannah.nongnu.org/projects/lwip/`

2.17 GPIO JTAG Recovery (gpio_jtag)

This example demonstrates changing the JTAG pins into GPIOs, a with a mechanism to revert them to JTAG pins. When first run, the pins remain in JTAG mode. Pressing the USR_SW1 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.

2.18 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.

Open a terminal with 115,200 8-N-1 to see the output for this demo.

2.19 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-TM4C1294XL 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.

2.20 Hibernate Calendar Mode Example (hibernate_calendar)

An example to demonstrate the use of the Hibernation module. The user can put the microcontroller in hibernation by typing ’hib’ in the terminal and pressing ENTER or by pressing USR_SW1 on the board. The microcontroller will then wake on its own after 5 seconds, or immediately if the user presses the RESET button. The External WAKE button, external WAKE pins, and GPIO (PK6) wake sources can also be used to wake immediately from hibernation. The following wiring enables the use of these pins as wake sources. WAKE on breadboard connection header (X11-95) to GND PK6 on BoosterPack 2 (X7-17) to GND PK6 on breadboard connection header (X11-63) to GND

The program keeps a count of the number of times it has entered hibernation. The value of the counter is stored in the battery-backed memory of the Hibernation module so that it can be retrieved when the microcontroller wakes. The program displays the wall time and date by making use of the calendar function of the Hibernate module. User can modify the date and time if so desired.

2.21 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 UART; GPIO pins B3, L1 and L0 (the GPIO on jumper J27 on the left edge of the board) 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).

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

2.23 Project Zero (project0)

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

2.24 PWM Reload Interrupt Demo (pwm_interrupt)

This example shows how to configure PWM2 for a load interrupt. The PWM interrupt will trigger every time the PWM2 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 F peripheral (for PWM2 pin)
- PWM2 - PF2

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

2.25 Sleep Modes(sleep_modes)

This example demonstrates the different power modes available on the Tiva C Series devices. The user button (USR-SW1) is used to cycle through the different power modes. The SRAM, Flash, and LDO are all configured to a lower power setting for the different modes.

A timer is configured to toggle an LED in an ISR in both Run and Sleep mode. In Deep-Sleep the PWM is used to toggle the same LED in hardware. The three remaining LEDs are used to indicate the current power mode.

LED key in addition to the toggling LED: 3 LEDs on - Run Mode 2 LEDs on - Sleep Mode 1 LED on - Deep-Sleep Mode

TODO: If using an IDE, disconnect from the Debug Session before attempting to enter Deep-Sleep or else the JTAG interface will conflict with entering Deep-Sleep mode and trigger a FaultISR!

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

2.26 SSI Master-to-Slave Transfer (spi_master_slave_xfer)

This example demonstrates how to configure SSI0 as a SSI Master and SSI1 as a SSI slave. The master will send four characters on the master to the slave using the legacy mode. In legacy mode, one bit is sent on each SSI Clock pulse. Once the SSI slave receives the four characters in the receive FIFO it will generate an interrupt.

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

- SSI0 peripheral
- GPIO Port A peripheral (for SSI0 pins)
- SSI0Clk - PA2
- SSI0Fss - PA3
- SSI0TX - PA4
- SSI0RX - PA5
- SSI1 peripheral
- GPIO Port B & E peripheral (for SSI1 pins)
- SSI1Clk - PB5
- SSI1Fss - PB4
- SSI1TX - PE4
- SSI1RX - PE5

This example requires board level connection between SSI0 and SSI1.

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

2.27 Quad-SSI Master (ssi_quad_mode)

This example shows how to configure SSI0 as a Quad-SSI Master and SSI1 as a Quad-SSI slave. The master device will send four characters to the slave device using the advanced Quad mode. In Quad-SSI mode, four bits are sent on each SSI Clock pulse. Once the Quad-SSI slave receives the four characters in its receive FIFO it will generate an interrupt.

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

- SSI0 peripheral
- GPIO Port A peripheral (for SSI0 pins)
- SSI0Clk - PA2
- SSI0Fss - PA3
- SSI0XDAT0 - PA4
- SSI0XDAT1 - PA5
- SSI0XDAT2 - PA6

- SSI0XDAT3 - PA7

- SSI1 peripheral
- GPIO Port B, D, E peripheral (for SSI1 pins)
- SSI1Clk - PB5
- SSI1Fss - PB4
- SSI1XDAT0 - PE4
- SSI1XDAT1 - PE5
- SSI1XDAT2 - PD4
- SSI1XDAT3 - PD5

This example requires board level connection between SSI0 and SSI1.

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

2.28 Timer Edge Capture Demo (timer_edge_capture)

This example demonstrates how to use two timers to determine the duration of the high period of an input signal. The same logic can be applied to also measure the low period. The example uses Timer 0 in Split Mode and the 8-bit prescaler to create two 24-bit wide timers. There are some limitations with this method to be aware of. The limitation on the lower end is if a signal is too fast to sample with the timers due to the time to execute code. The limitation on the upper end is if the signal period is longer than 139.8 milliseconds at which point the 24-bit timer would overflow. That issue can be worked around by using two 32-bit timers which would allow measuring a signal up to 35.79 seconds long.

To test this example either input a square wave to PL4 and PL5 with a signal generator or leverage a second LaunchPad running a TivaWare PWM project such as pwm_interrupt.

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

- GPIO Port L peripheral (for TIMER0 pins)
- T0CCP0 - PL4
- T0CCP1 - PL5

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

2.29 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 throught the UART.

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

2.30 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.

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

2.32 uDMA Scatter Gather Demo (udma_scatter_gather)

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 channel using the scatter-gather mode. The test runs for 10 seconds before exiting.

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

2.33 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 "TivaWare for C Series PC Companion Utilities" package (SW-TM4C-USB-WIN) on the installation CD or via download from `http://www.ti.com/tivaware`. Project files are included to allow the examples to be built using Microsoft Visual Studio 2008. Source code for this application can be found in directory ti/TivaWare_C_Series-x.x/tools/usb_bulk_example.

2.34 USB CDC Serial Device example (usb_dev_cdcserial)

This example application turns the EK-TM4C1294XL LaunchPad into a single port USB CDC device when connected to a USB host system. The application supports the USB Communication Device Class, Abstract Control Model to by interacting with the USB host via a virtual COM port. For this example, the evaluation kit will enumerate as a CDC device with one virtual serial port that redirects UART0 traffic to and from the USB host system.

The virtual serial port will echo data to the physical UART0 port on the device which is connected to the virtual serial port on the ICDI device on this board. The physical UART0 will also echo onto the virtual serial device provided by the TM4C controller.

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.

2.35 USB Composite Serial Device (usb_dev_cserial)

This example application turns the evaluation kit into a multiple virtual serial ports 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. For this example, the evaluation kit will enumerate as a composite device with two virtual serial ports. Including the physical UART0 connection with the ICDI, this means that three independent virtual serial ports will be visible to the USB host.

The first virtual serial port will echo data to the physical UART0 port on the device which is connected to the virtual serial port on the ICDI device on this board. The physical UART0 will also echo onto the first virtual serial device provided by the TM4C controller.

The second TM4C virtual serial port will provide a console that can echo data to both the ICDI virtual serial port and the first TM4C virtual serial port. It will also allow turning on, off or toggling the boards led status. Typing a "?" and pressing return should echo a list of commands to the terminal, since this board can show up as possibly three individual virtual serial devices.

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.

2.36 USB HID Keyboard Device (usb_dev_keyboard)

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

The device implemented by this application also supports USB remote wakeup allowing it to request

the host to reactivate a suspended bus. If the bus is suspended (as indicated on the terminal window), pressing the push button will request a remote wakeup assuming the host has not specifically disabled such requests.

2.37 USB HID Keyboard Host (usb_host_keyboard)

This application demonstrates the handling of a USB keyboard attached to the evaluation kit. Once attached, text typed on the keyboard will appear on the UART. Any keyboard that supports the USB HID BIOS protocol is supported.

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

2.38 USB Host mouse example(usb_host_mouse)

This example application demonstrates how to support a USB mouse using the EK-TM4C129X evaluation kit. This application supports only a standard mouse HID device.

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

2.39 USB Mass Storage Class Host (usb_host_msc)

This example application demonstrates reading a file system from a USB mass storage class device. It makes use of FatFs, a FAT file system driver. It provides a simple command console via the UART for issuing commands to view and navigate the file system on the mass storage device.

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

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

2.40 USB Stick Update Demo (usb_stick_demo)

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

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

This program simply displays a message on the screen and prompts the user to press the USR_SW1 button. Once the button is pressed, control is passed back to the usb_stick_update program which is still is flash, and it will attempt to load another program from the memory stick. This shows how a user application can force a new firmware update from the memory stick.

2.41 USB Memory Stick Updater (usb_stick_update)

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

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

- no user application is present at **APP_START_ADDRESS**
- the user application has requested an update by transferring control to the updater
- the user holds down the USR_SW1 button when the board is reset

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

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

2.42 Watchdog (watchdog)

This example application demonstrates the use of the watchdog as a simple heartbeat for the system. If the watchdog is not periodically fed, it will reset the system. Each time the watchdog is fed, the LED is inverted so that it is easy to see that it is being fed, which occurs once every second. To stop the watchdog being fed and, hence, cause a system reset, press the SW1 button.

3 Buttons Driver

3.1 Introduction

The buttons driver provides functions to make it easy to use the push buttons on this 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-tm4c1294xl/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 called 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.

```
//
// Map Left button to the GPIO Pin 0 of the button port.
//
#define LEFT_BUTTON             GPIO_PIN_0

//
// The button example
//
void
ButtonExample(void)
{
    unsigned char ucDelta, ucState;

    //
    // 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(LEFT_BUTTON, ucState, ucDelta))
            {
                //
                // TODO: SELECT button action code
                //
            }
        }
    }
```

4 Pinout Module

4.1 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/ek-tm4c1294xl/drivers`, with `pinout.c` containing the source code and `pinout.h` containing the API declarations for use by applications.

4.2 API Functions

Functions

- void LEDRead (uint32_t *pui32LEDValue)
- void LEDWrite (uint32_t ui32LEDMask, uint32_t ui32LEDValue)
- void PinoutSet (bool bEthernet, bool bUSB)

4.2.1 Function Documentation

4.2.1.1 LEDRead

This function reads the state to the LED bank.

Prototype:

```
void
LEDRead(uint32_t *pui32LEDValue)
```

Parameters:

pui32LEDValue is a pointer to where the LED value will be stored.

Description:

This function reads the state of the CLP LEDs and stores that state information into the variable pointed to by pui32LEDValue.

Returns:

None.

4.2.1.2 LEDWrite

This function writes a state to the LED bank.

Prototype:

```
void
LEDWrite(uint32_t ui32LEDMask,
         uint32_t ui32LEDValue)
```

Parameters:

ui32LEDMask is a bit mask for which GPIO should be changed by this call.

ui32LEDValue is the new value to be applied to the LEDs after the ui32LEDMask is applied.

Description:

The first parameter acts as a mask. Only bits in the mask that are set will correspond to LEDs that may change. LEDs with a mask that is not set will not change. This works the same as GPIOPinWrite. After applying the mask the setting for each unmasked LED is written to the corresponding LED port pin via GPIOPinWrite.

Returns:

None.

4.2.1.3 PinoutSet

Configures the device pins for the standard usages on the EK-TM4C1294XL.

Prototype:

```
void
PinoutSet(bool bEthernet,
          bool bUSB)
```

Parameters:

bEthernet is a boolean used to determine function of Ethernet pins. If true Ethernet pins are configured as Ethernet LEDs. If false GPIO are available for application use.

bUSB is a boolean used to determine function of USB pins. If true USB pins are configured for USB use. If false then USB pins are available for application use as GPIO.

Description:

This function enables the GPIO modules and configures the device pins for the default, standard usages on the EK-TM4C1294XL. 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.

4.3 Programming Example

The following example shows how to configure the device pins.

```
//
// The pinout example.
//
void
PinoutExample(void)
{
    //
    // Configure the device pins.
    // First argument determines whether the Ethernet pins will be configured
    // in networking mode for this application.
    // Second argument determines whether the USB pins will be configured for
    // USB mode for this application.
    //
    PinoutSet(true, false);
}
```

5 HTTP Driver

5.1 Introduction

The http.c file provides functions to facilatate the creation and management of HTTP application. These functions are used by the Exosite Abstraction Layer `exosite_hal_lwip.c` to send HTTP requests to the Exosite server for IoT applications.

This driver is located in `examples/boards/ek-tm4c1294xl/drivers`, with `http.c` containing the source code and `http.h` containing the API declarations for use by applications.

5.2 API Functions

Functions

- void HTTPMessageBodyAdd (char *pcDest, char *pcBodyData)
- void HTTPMessageHeaderAdd (char *pcDest, char *pcHeaderName, char *pcHeaderValue)
- void HTTPMessageTypeSet (char *pcDest, uint8_t ui8Type, char *pcResource)
- void HTTPResponseBodyExtract (char *pcData, char *pcDest)
- void HTTPResponseHeaderExtract (char *pcData, uint32_t ui32HeaderIdx, char *pcHeaderName, char *pcHeaderValue)
- uint32_t HTTPResponseParse (char *pcData, char *pcResponseText, uint32_t *pui32NumHeaders)

5.2.1 Function Documentation

5.2.1.1 HTTPMessageBodyAdd

Add body data to to a HTTP request.

Prototype:

```
void
HTTPMessageBodyAdd(char *pcDest,
                   char *pcBodyData)
```

Parameters:

pcDest is a pointer to the destination/output string.

pcBodyData is a pointer to a string containing the body data. This can be anything from HTML to encoded data (such as JSON).

Description:
Note that this function must be called after HTTPMessageTypeSet() and HTTPMessageHeaderAdd() as it simply appends the body data to an existing string/buffer.

Returns:
None.

5.2.1.2 HTTPMessageHeaderAdd

Add a header to a HTTP request.

Prototype:
```
void
HTTPMessageHeaderAdd(char *pcDest,
                     char *pcHeaderName,
                     char *pcHeaderValue)
```

Parameters:
pcDest is a pointer to the destination/output string.
pcHeaderName is a pointer to a string containing the header name.
pcHeaderValue is a pointer to a string containing the header data.

Description:
Note that this function must be called after HTTPMessageTypeSet() as it simply appends a header to an existing string/buffer.

Returns:
None.

5.2.1.3 HTTPMessageTypeSet

Set the HTTP message type.

Prototype:
```
void
HTTPMessageTypeSet(char *pcDest,
                   uint8_t ui8Type,
                   char *pcResource)
```

Parameters:
pcDest is a pointer to the destination/output string.
ui8Type is the HTTP request type. Macros such as HTTP_MESSAGE_GET are defined in http.h.
pcResource is a pointer to a string containing the resource portion of the HTTP message. The resource goes in between the type (ex: GET) and HTTP suffix on the first line of a HTTP request. An example would be index.html.

Description:
This function should be called to start off a new HTTP request.

Returns:
None.

5.2.1.4 HTTPResponseBodyExtract

Extract the body from a HTTP response string/buffer.

Prototype:

```
void
HTTPResponseBodyExtract(char *pcData,
                        char *pcDest)
```

Parameters:

pcData is a pointer to the source string/buffer.

pcDest is a pointer to a string that will receive the body data.

Returns:

None.

5.2.1.5 void HTTPResponseHeaderExtract (char * *pcData*, uint32_t *ui32HeaderIdx*, char * *pcHeaderName*, char * *pcHeaderValue*)

Extract a specified header from a HTTP response string/buffer.

Parameters:

pcData is a pointer to the source string/buffer.

ui32HeaderIdx specifies the index of the header to extract.

pcHeaderName is a pointer to a string that will receive the name of the header specified by ui32HeaderIdx.

pcHeaderValue is a pointer to a string that will receive the value of the header specified by ui32HeaderIdx.

Description:

Note that this function should be used in conjunction with HTTPResponseParse() since it notifies the application of the number of headers in a string/buffer.

Returns:

None.

5.2.1.6 HTTPResponseParse

Parse a HTTP response.

Prototype:

```
uint32_t
HTTPResponseParse(char *pcData,
                  char *pcResponseText,
                  uint32_t *pui32NumHeaders)
```

Parameters:

pcData is a pointer to the source string/buffer.

pcResponseText is a pointer to a string that will receive the response text from the first line of the HTTP response.

pui32NumHeaders is a pointer to a variable that will receive the number of headers detected in pcData.

Description:

Note that this function must be called after HTTPMessageTypeSet() as it simply appends a header to an existing string/buffer.

Returns:

Returns the HTTP response code. If parsing error occurs, returns 0.

6 Ethernet Client Driver

6.1 Introduction

The eth_client_lwip.c file provides Ethernet client functions to interface with the server. These functions are used by the Exosite Abstraction Layer `exosite_hal_lwip.c` to manage connection with the Exosite server in IoT applications.

This driver is located in `examples/boards/ek-tm4c1294xl/drivers`, with `eth_client_lwip.c` containing the source code and `eth_client_lwip.h` containing the API declarations for use by applications.

6.2 API Functions

Functions

- uint32_t EthClientAddrGet (void)
- err_t EthClientDHCPConnect (void)
- int32_t EthClientDNSResolve (void)
- void EthClientHostSet (const char *pcHostName, uint16_t ui16Port)
- void EthClientInit (uint32_t ui32SysClock, tEventFunction pfnEvent)
- void EthClientMACAddrGet (uint8_t *pui8MACAddr)
- void EthClientProxySet (const char *pcProxyName, uint16_t ui16Port)
- int32_t EthClientSend (int8_t *pi8Request, uint32_t ui32Size)
- uint32_t EthClientServerAddrGet (void)
- int32_t EthClientTCPConnect (void)
- void EthClientTCPDisconnect (void)
- void lwIPHostTimerHandler (void)
- err_t TCPConnected (void *pvArg, struct tcp_pcb *psPcb, err_t iErr)
- void TCPError (void *vPArg, err_t iErr)
- err_t TCPReceived (void *pvArg, struct tcp_pcb *psPcb, struct pbuf *psBuf, err_t iErr)
- err_t TCPSent (void *pvArg, struct tcp_pcb *psPcb, u16_t ui16Len)

6.2.1 Function Documentation

6.2.1.1 EthClientAddrGet

Returns the IP address for this interface.

Prototype:

```
uint32_t
EthClientAddrGet(void)
```

Description:

This function will read and return the currently assigned IP address for the Tiva Ethernet interface.

Returns:

Returns the assigned IP address for this interface.

6.2.1.2 EthClientDHCPConnect

DHCP connect

Prototype:

```
err_t
EthClientDHCPConnect(void)
```

Description:

This function obtains the MAC address from the User registers, starts the DHCP timer and blocks until an IP address is obtained.

Returns:

This function always returns ERR_OK.

6.2.1.3 EthClientDNSResolve

Handler function when the DNS server gets a response or times out.

Prototype:

```
int32_t
EthClientDNSResolve(void)
```

Description:

This function is called when the DNS server resolves an IP or times out. If the DNS server returns an IP structure that is not NULL, add the IP to to the g_sEnet.sResolvedIP IP structure.

Returns:

This function will return an lwIP defined error code.

6.2.1.4 EthClientHostSet

Set the host string for the Ethernet connection.

Prototype:

```
void
EthClientHostSet(const char *pcHostName,
                 uint16_t ui16Port)
```

Parameters:

pcHostName is the string used as the host server name.

ui16Port is the string used as the host port.

Description:

This function sets the current host used by the Ethernet connection. The *pcHostName* value can be 0 to indicate that no host is in use or it can be a pointer to a string that holds the name of the host server to use. The content of the pointer passed to *pcHostName* should not be changed after this call as this function only stores the pointer and does not copy the data from this pointer.

Returns:

None.

6.2.1.5 EthClientInit

Initialize the Ethernet client

Prototype:

```
void
EthClientInit(uint32_t ui32SysClock,
              tEventFunction pfnEvent)
```

Parameters:

ui32SysClock is the input for the system clock setting of the TM4C

pfnEvent is the network event handler.

Description:

This function initializes all the Ethernet components to not configured. This tells the SysTick interrupt which timer modules to call.

Returns:

None.

6.2.1.6 EthClientMACAddrGet

Returns the MAC address for the Tiva Ethernet controller.

Prototype:

```
void
EthClientMACAddrGet(uint8_t *pui8MACAddr)
```

Parameters:

pui8MACAddr is the 6 byte MAC address assigned to the Ethernet controller.

Description:

This function will read and return the MAC address for the Ethernet controller.

Returns:

Returns the weather server IP address for this interface.

6.2.1.7 EthClientProxySet

Set the proxy string for the Ethernet connection.

Prototype:

```
void
EthClientProxySet(const char *pcProxyName,
                  uint16_t ui16Port)
```

Parameters:

pcProxyName is the string used as the proxy server name.

ui16Port is the string used as the proxy port.

Description:

This function sets the current proxy used by the Ethernet connection. The *pcProxyName* value can be 0 to indicate that no proxy is in use or it can be a pointer to a string that holds the name of the proxy server to use. The content of the pointer passed to *pcProxyName* should not be changed after this call as this function only stores the pointer and does not copy the data from this pointer.

Returns:

None.

6.2.1.8 EthClientSend

Send a request to the server

Prototype:

```
int32_t
EthClientSend(int8_t *pi8Request,
              uint32_t ui32Size)
```

Parameters:

pi8Request request to be sent

ui32Size length of the request to be sent. This is usually the size of the request minus the termination character.

Description:

This function will send the request to the connected server

Returns:

This function will return an lwIP defined error code.

6.2.1.9 EthClientServerAddrGet

Returns the weather server IP address for this interface.

Prototype:

```
uint32_t
EthClientServerAddrGet(void)
```

Description:
This function will read and return the server IP address that is currently in use. This could be the proxy server if the Internet proxy is enabled.

Returns:
Returns the weather server IP address for this interface.

6.2.1.10 EthClientTCPConnect

TCP connect

Prototype:
```
int32_t
EthClientTCPConnect(void)
```

Description:
This function attempts to connect to a TCP endpoint.

Returns:
This functions returns a '0' if successful, or a '1' if an error an error has occurred.

6.2.1.11 EthClientTCPDisconnect

TCP discconnect

Prototype:
```
void
EthClientTCPDisconnect(void)
```

Description:
This function attempts to disconnect a TCP endpoint.

Returns:
None.

6.2.1.12 lwIPHostTimerHandler

Periodic Tick for the Ethernet client

Prototype:
```
void
lwIPHostTimerHandler(void)
```

Description:
This function is the needed periodic tick for the Ethernet client. It needs to be called periodically through the use of a timer or systick.

Returns:
None.

6.2.1.13 TCPConnected

Finalizes the TCP connection in client mode.

Prototype:

```
static err_t
TCPConnected(void *pvArg,
             struct tcp_pcb *psPcb,
             err_t iErr)
```

Parameters:

pvArg is the state data for this connection.

psPcb is the pointer to the TCP control structure.

iErr is not used in this implementation.

Description:

This function is called when the lwIP TCP/IP stack has completed a TCP connection.

Returns:

This function will return an lwIP defined error code.

6.2.1.14 TCPError

Handles lwIP TCP/IP errors.

Prototype:

```
static void
TCPError(void *vPArg,
         err_t iErr)
```

Parameters:

vPArg is the state data for this connection.

iErr is the error that was detected.

Description:

This function is called when the lwIP TCP/IP stack has detected an error. The connection is no longer valid.

Returns:

None.

6.2.1.15 TCPReceived

Finalizes the TCP connection in client mode.

Prototype:

```
err_t
TCPReceived(void *pvArg,
            struct tcp_pcb *psPcb,
            struct pbuf *psBuf,
            err_t iErr)
```

Parameters:

pvArg is the state data for this connection.

psPcb is the pointer to the TCP control structure.

psBuf

iErr is not used in this implementation.

Description:

This function is called when the lwIP TCP/IP stack has completed a TCP connection.

Returns:

This function will return an lwIP defined error code.

6.2.1.16 TCPSent

Handles acknowledgment of data transmitted via Ethernet.

Prototype:

```
static err_t
TCPSent(void *pvArg,
        struct tcp_pcb *psPcb,
        u16_t ui16Len)
```

Parameters:

pvArg is the state data for this connection.

psPcb is the pointer to the TCP control structure.

ui16Len is the length of the data transmitted.

Description:

This function is called when the lwIP TCP/IP stack has received an acknowledgment for data that has been transmitted.

Returns:

This function will return an lwIP defined error code.

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
Copyright © 2013-2020, Texas Instruments Incorporated