Application Note

Developing Advanced Applications With FreeRTOS on TM4C MCUs

Ralph Jacobi and Charles Tsai

ABSTRACT

FreeRTOS is a real-time operating system for embedded systems. It has been widely ported to many architecture platforms due to its compact size and being distributed under free open source licensing. This application report is a continuation of the application report *Developing Common Applications With FreeRTOS on TM4C MCUs* and provides additional examples for the USB and Ethernet peripherals to demonstrate how to use the FreeRTOS kernel on the Texas Instruments TM4C family of Arm® Cortex®-M4F microcontrollers.

The source code for all example projects discussed in this application report can be downloaded from the following link http://www.ti.com/lit/zip/spma087.

Table of Contents

Trademarks

1 Introduction

The microcontroller market continues to see a steady increase in the utilization of embedded real-time operating systems (RTOS) to manage applications. The increased focus and need for RTOS solutions has led to a number of open source RTOS offerings including the widely used FreeRTOS. To address the growing need for FreeRTOS support on the TM4C Arm Cortex-M4F microcontroller series, a series of application reports have been released. *Developing Basic Applications With FreeRTOS on TM4C MCU* was the initial release and provides a set of fundamental application examples. *Developing Common Applications With FreeRTOS on TM4C MCUs* followed to expand the application coverage to highlight many commonly used TM4C peripherals. This final application report in the series will cover the more advanced USB and Ethernet peripherals.

Included with this application report are example projects for both the TM4C123x and the TM4C129x device families. These projects are based on the bare metal examples provided in the TivaWare Software Development Kit (SDK). The TivaWare™ SDK includes Driver Library (DriverLib) APIs for all peripherals that are the building blocks for any application on a TM4C microcontroller. The provided example projects demonstrate how to use DriverLib APIs within the FreeRTOS kernel to create simple real-world applications with various device peripherals. The peripherals that are demonstrated in this report are as follows.

- Universal Serial Bus (USB)
- Ethernet

2 How to Install

The following steps demonstrate how to download the latest version of TivaWare and FreeRTOS. This section can be skipped if it was done previously.

1. Download the latest TivaWare SDK from here and follow the installer instructions. If using the default installation path, the TivaWare SDK will be installed at `C:\ti\TivaWare_C_Series-2.2.0.295`. For the TivaWare directory structure, see Figure 2-1 (and notice there is already a `third_party/FreeRTOS directory`). The existing FreeRTOS installation in the TivaWare library is version 8.2.3. This needs to be updated to the latest version.

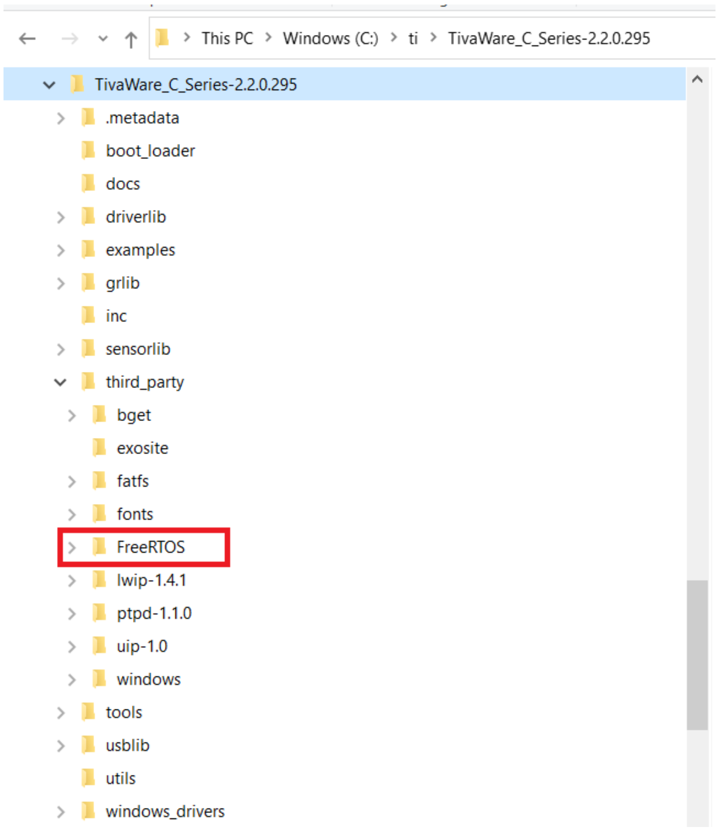
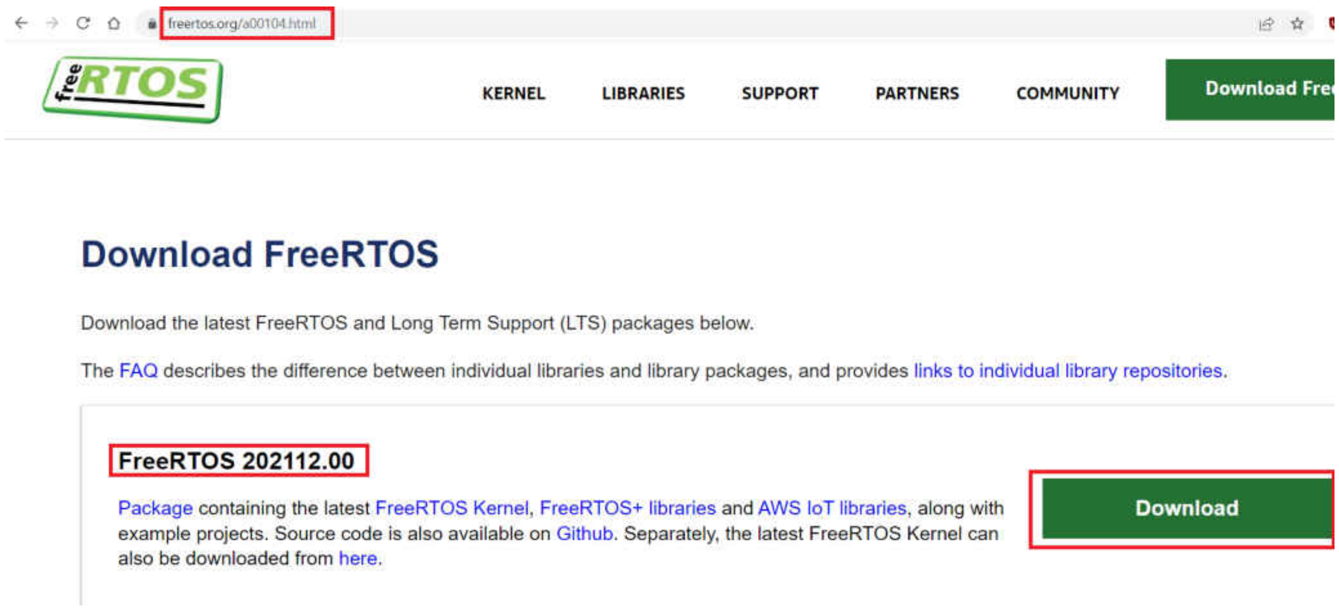


Figure 2-1. TivaWare Directory Structure

2. Download the latest version of FreeRTOS from the official download source here. At the time of the publishing of this application report, the latest version of FreeRTOS is Version 202112.00, as shown in Figure 2-2. All examples provided with this application report have been created and validated using that version.



Download the latest FreeRTOS and Long Term Support (LTS) packages below.

The FAQ describes the difference between individual libraries and library packages, and provides links to individual library repositories.

Package containing the latest FreeRTOS Kernel, FreeRTOS+ libraries and AWS IoT libraries, along with example projects. Source code is also available on Github. Separately, the latest FreeRTOS Kernel can also be downloaded from here.

Figure 2-2. FreeRTOS Download Site

3. Unzip the FreeRTOS zip file and install to a temporary directory. View the FreeRTOS directory structure and notice the FreeRTOS folder and other folders and files, as shown in Figure 2-3.

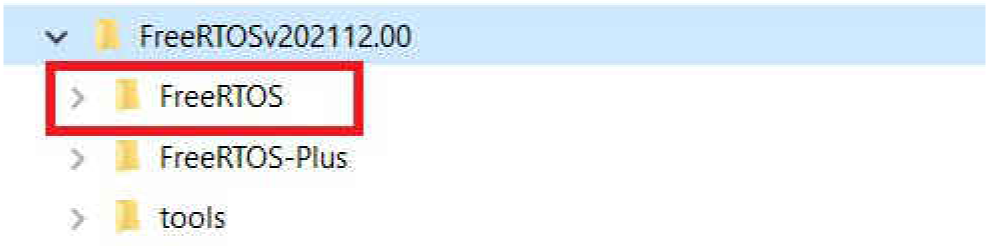


Figure 2-3. FreeRTOS version 202112.00 Directory Structure

2.1 Update the FreeRTOS Version in the TivaWare Directory

The FreeRTOS folder from the new download will need to be either moved or copied to the TivaWare library under `C:\ti\TivaWare_C_Series-2.2.0.295\third_party`. Make sure to first rename or move the existing FreeRTOS folder in the TivaWare library if there is a desire to retain it as a backup as shown in Figure 2-4.

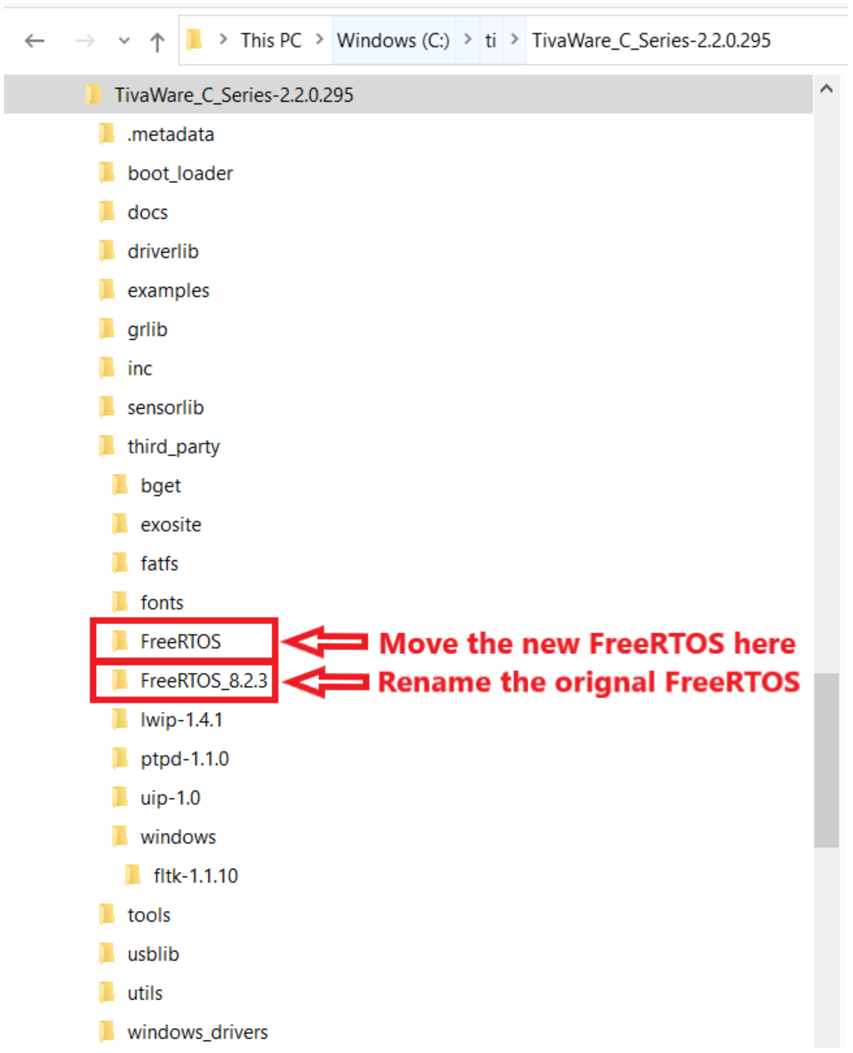


Figure 2-4. Updated TivaWare for New FreeRTOS Installation

2.2 Adding FreeRTOS Hardware Driver Files for TM4C LaunchPads

The collateral provided include new hardware driver files that were created to provide a single source for basic pinout configurations for each LaunchPad Evaluation Kit. The files are located under the `Driver Files` directory. These files are referenced in all examples that use device pins, so this step is required as part of the first time setup.

The *rtos_hw_drivers.c* and *rtos_hw_drivers.h* files for each specific LaunchPad should be copied into the TivaWare folders for that LaunchPad under the `drivers` directory. For EK-TM4C123GXL this path would be `TivaWare_C_Series-2.2.0.295\examples\boards\ek-tm4c123gxl\drivers` and for EK-TM4C1294XL this path would be `TivaWare_C_Series-2.2.0.295\examples\boards\ek-tm4c1294xl\drivers`.

3 Architecture for TM4C FreeRTOS Examples

The provided example projects are designed with the purpose of offering a compact and streamlined foundation that can be used to begin application development. In order to achieve this, the FreeRTOS kernel is used in conjunction with the TivaWare Driver Library (DriverLib) without any additional abstraction layers added. The configuration and handling of specific peripherals are contained within dedicated task files, which allow for easy re-use across multiple projects.

At this time, POSIX is not included as part of the architecture. The FreeRTOS + POSIX solution currently is a partial implementation that is only offered via GitHub. Details about that offering can be read about at this link.

In addition to DriverLib, TivaWare also includes some utilities to complement specific applications such as outputting UART messages over a terminal or using external memory sources. Be aware that these utilities were created for bare metal applications and may include APIs that were not designed to be thread safe. For the simplicity of demonstrations, there are times certain non-thread safe APIs are used in example projects. These are clearly highlighted as non-thread safe, and they are only used to demonstrate what is being executed as part of the example project.

3.1 Proper Clock Configuration

To ensure that the FreeRTOS kernel is operating at the correct tick rate, care must be taken to assign the correct system clock frequency of the TM4C microcontroller to the FreeRTOS kernel. The *configCPU_CLOCK_HZ* variable in `FreeRTOSConfig.h` holds the value of the system clock frequency. This variable must be updated accordingly for any system clock frequency changes. All examples provided are using the maximum supported clock speed for the target microcontroller which is 80 MHz for TM4C123x devices and 120 MHz for TM4C129x devices.

The device-specific data sheets for each TM4C microcontroller provides information about the valid clock speed options under the *Clock Control* section. There are limitations to how precise a frequency can be set based on the clock dividers. If a frequency that cannot be supported is input in a TivaWare clock configuration function, typically the next closest frequency will be configured. Improper clock configurations in terms of not accurately reflecting external crystals or which oscillator should be used can also lock a device until reset to factory condition. Lastly, certain peripherals have requirements for minimum system clock frequencies to be able to operate to specification including the ADC (for high speed sampling), USB, and Ethernet (TM4C129x family only) peripherals.

There are differences in how to handle the clock configuration between the two TM4C device families.

For TM4C123x devices, the system clock frequency is set with the `SysCtlClockSet` API and is determined only by the divider that is passed with the function call. The actual rate of the system clock can be then acquired with the `SysCtlClockGet` API. However, since the `FreeRTOSConfig.h` file requires hardcoded definitions of the clock rate, any changes to the system clock frequency will need to be validated without the kernel running before plugging in the new value for *configCPU_CLOCK_HZ*.

For TM4C129x devices, the system clock frequency is set with the `SysCtlClockFreqSet` API, which requires an input for the desired system clock frequency. It then returns the actual system clock frequency that was set based on what is the closest setting to the requested value. In each example provided, that value is stored in *g_ui32SysClock*, which can be used to verify that the correct frequency has been set. The clock frequency input provided to `SysCtlClockFreqSet` is the *configCPU_CLOCK_HZ* variable. Therefore, as long as a valid TM4C129x clock frequency is defined, only one change is needed for the project.

3.2 How to use Hardware Interrupts Alongside the FreeRTOS Kernel

An important element of managing an RTOS on a microcontroller is correctly handling the use of hardware interrupts from the microcontroller without disrupting the RTOS kernel and scheduler. Because FreeRTOS aims to be chip and compiler agonistic, there is no unique plug-in for device specific hardware interrupts. Instead, hardware interrupt processing occurs outside of the kernel and scheduler and must be handled by using very lean interrupt service routines (ISRs). Minimizing the cycles spent executing the ISR allows the scheduler to take control of the application quicker.

Because most ISRs require the processing of a specific event that has been triggered, the FreeRTOS kernel provides methods to defer the processing of events from an ISR to a task that is part of the kernel and runs according to the scheduler. These deferred tasks have configured priorities that allow developers the same flexibility as ISR processing where priorities can be set to verify that the most important interrupts are processed first. With this method, the hardware ISR processing can be minimalized while still retaining priorities as required by an application.

To learn more details about deferred interrupt processing as well as FreeRTOS interrupt safe API and how context switching is performed by the kernel, see the *Interrupt Management* chapter in Mastering the FreeRTOS Real Time Kernel.

From a TM4C microcontroller perspective, configuring interrupts includes mapping the ISR to the interrupt vector table so that the ISR is executed whenever the hardware interrupt is triggered. There are two methods to correctly register a new interrupt to the interrupt vector table. These are both documented in full detail in the *TivaWare Vector Tables and IntDefaultHandler* section of the *Getting Started with TivaWare™ User's Guide*.

3.3 Code Composer Studio Debug Support for FreeRTOS

The debug features that are available in Code Composer Studio when using the firmware provided in this application report is limited to traditional bare metal debug methods. Advanced RTOS specific methods including Real-Time Object View (ROV) are not supported on TM4C microcontrollers at this time. Access to such features typically comes with a full Software Development Kit (SDK) release which typically provide additional drivers, broader IDE support, and various tools to comprehensively evaluate a microcontroller.

It is possible to use third-party tools that are capable of interfacing with the FreeRTOS kernel to get information including stack usage of tasks and the state of queues, semaphore and so on. However, there is no officially supported solution for the combination of FreeRTOS and TM4C microcontrollers at this time. If full ROV support is required for a TM4C microcontroller, an alternative RTOS option would be to use TI-RTOS for Tiva-C. With TI-RTOS, the ROV Classic tool in Code Composer Studio can be used with TM4C microcontrollers.

4 Example Project Walkthroughs

This application report includes a zip file containing example projects for the EK-TM4C123GXL LaunchPad Evaluation Kit and EK-TM4C1294XL LaunchPad Evaluation Kit. . The Code Composer Studio™ projects demonstrate how to create applications on TM4C microcontrollers by using the TivaWare SDK with the FreeRTOS kernel. They are focused on demonstrating fundamental FreeRTOS features and basic Arm microcontroller peripherals. Table 4-1 shows a list of examples.

Table 4-1. FreeRTOS Application Examples

TM4C129	TM4C123
usb_dev_bulk	usb_dev_bulk
usb_dev_cdcserial	usb_dev_cdcserial
usb_dev_keyboard	usb_dev_keyboard
enet_lwip	
enet_io	

4.1 Download and Import the Examples

The collateral provided with this application report can either be unzipped into a local directory or kept in the zip format. Both formats can be imported to the CCS.

1. To import the project into CCS, first select the “File” -> ”Import”.

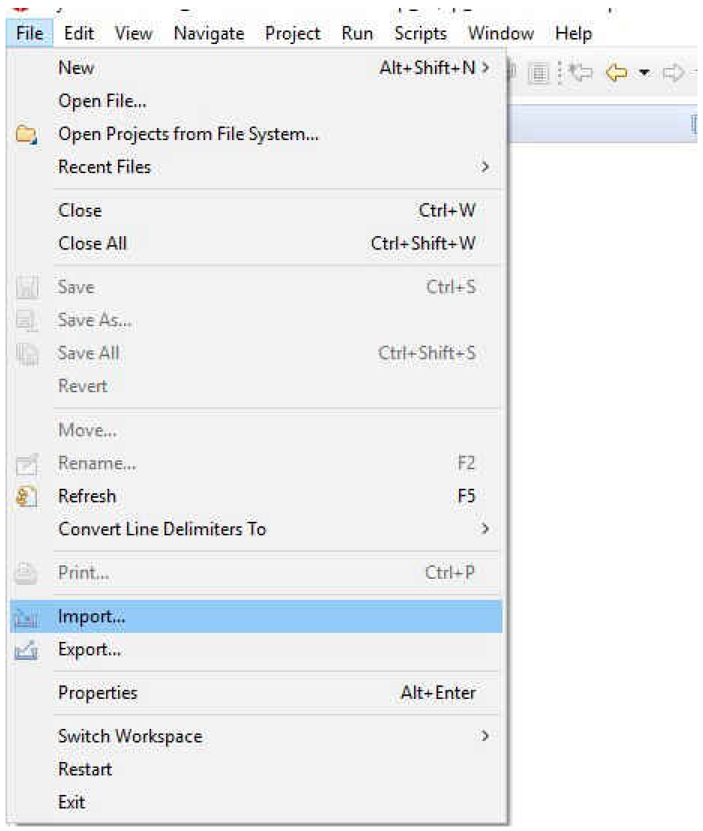
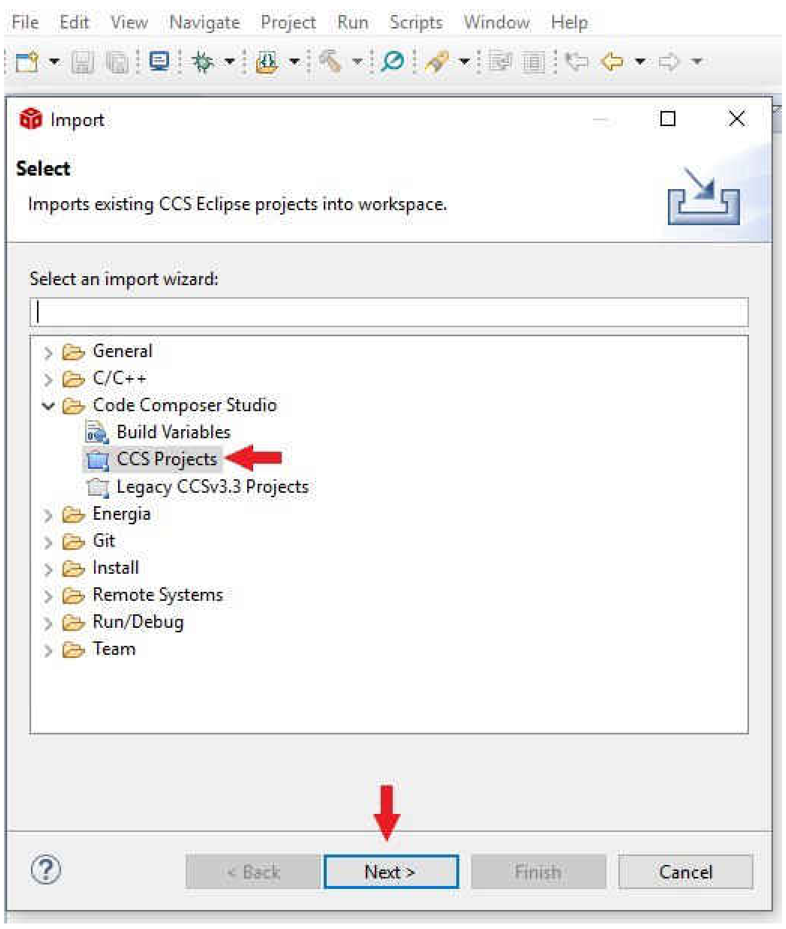


Figure 4-1. Import CCS Projects Step 1

2. Select "CCS Projects" to import the examples and then click "Next".



Select

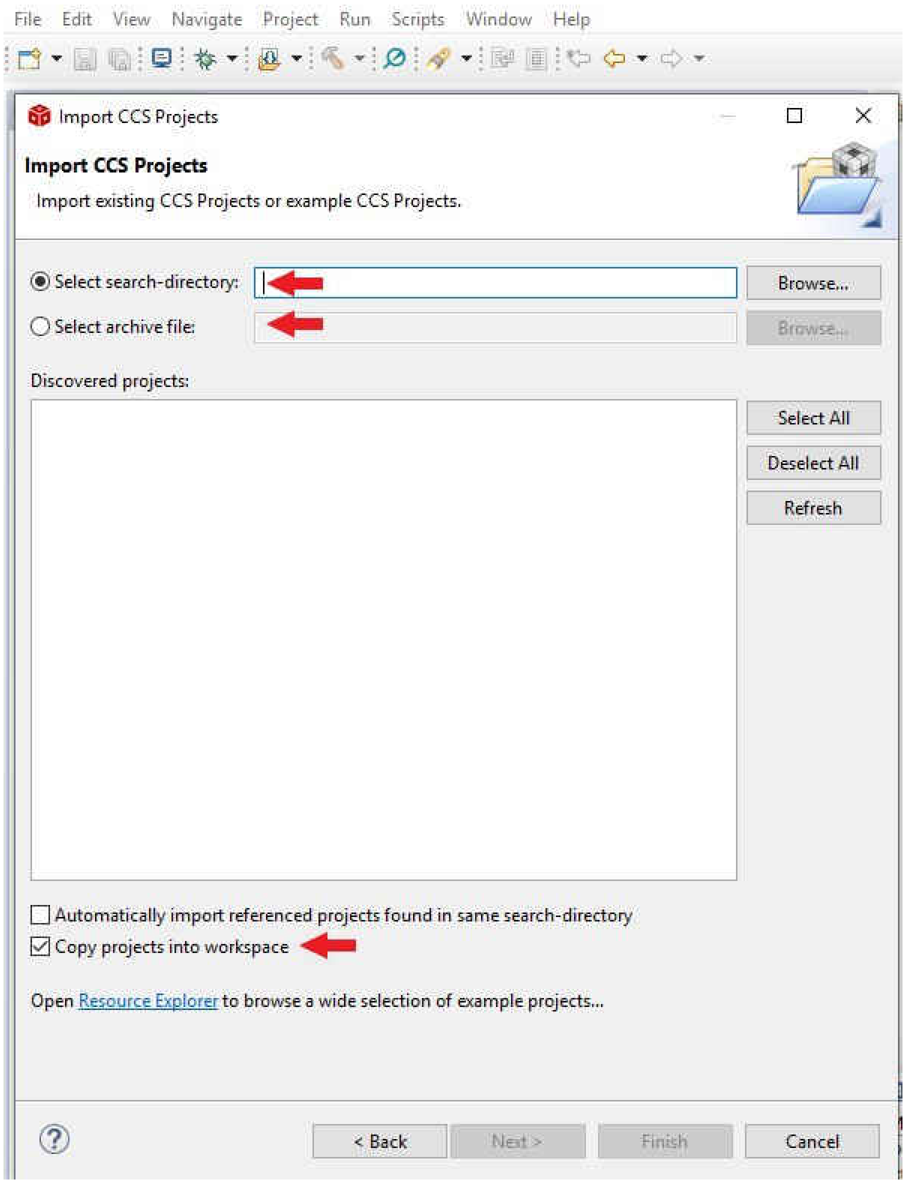
Imports existing CCS Eclipse projects into workspace.

Select an import wizard:

- General
- C/C++
- Code Composer Studio
  - Build Variables
  - CCS Projects
  - Legacy CCSv3.3 Projects
- Energia
- Git
- Install
- Remote Systems
- Run/Debug
- Team

Figure 4-2. Import CCS Projects Step 2

3. Next, provide the path to either the unzipped project by selecting the first radio button or import the zip file directly by selecting the second radio button. Click the “Copy projects into workspace”.



Import CCS Projects

Import existing CCS Projects or example CCS Projects.

(●) Select search-directory:

( ) Select archive file:

Discovered projects:

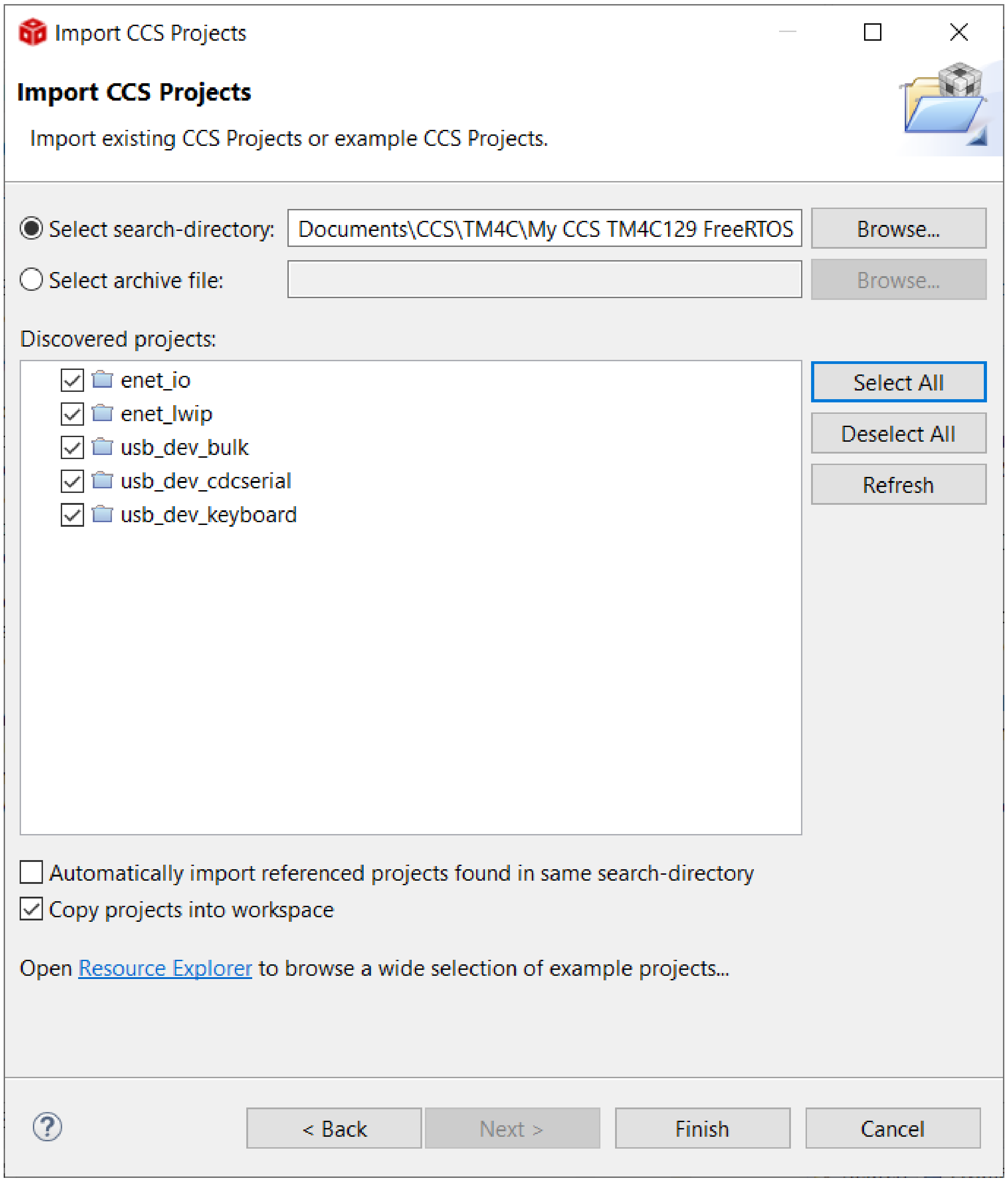
☐ Automatically import referenced projects found in same search-directory

☑ Copy projects into workspace

Open Resource Explorer to browse a wide selection of example projects...

Figure 4-3. Import CCS Projects Step 3

4. After the project path is provided, a total of five discovered projects will show up for TM4C129. First click "Select All" button and then click the "Finish" button to complete the import.


Import CCS Projects

Import CCS Projects

Import existing CCS Projects or example CCS Projects.

(●) Select search-directory: Documents\CCS\TM4C\My CCS TM4C129 FreeRTOS

( ) Select archive file:

Discovered projects:

- [✓] enet_io
- [✓] enet_lwip
- [✓] usb_dev_bulk
- [✓] usb_dev_cdcserial
- [✓] usb_dev_keyboard

[ ] Automatically import referenced projects found in same search-directory

[✓] Copy projects into workspace

Open Resource Explorer to browse a wide selection of example projects...

Figure 4-4. Import CCS Project Step 4

4.2 USB Examples

Most TM4C microcontrollers include a USB peripheral, but the functionality varies across devices. For TM4C129x devices, all devices have USB 2.0 Full Speed host and device support with the possibility to use USB 2.0 High Speed with an external USB PHY. For TM4C123x devices, the variant selected determines if there is no USB peripheral, USB 2.0 Full Speed device mode-only support, or USB 2.0 Full Speed host and device support.

Both TM4C LaunchPads use devices that have USB 2.0 Full Speed host and device support, but the EK-TM4C123GXL LaunchPad requires hardware modifications to use USB host mode that are documented in *Using USB Host Mode on the EK-TM4C123GXL LaunchPad*. To access the USB peripheral, either a second cable has to be connected to the secondary USB port on a LaunchPad, or the power configuration has to be adjusted to power the board through USB, which is done with a switch on the EK-TM4C123GXL LaunchPad and via a jumper on JP1 for the EK-TM4C1294XL LaunchPad.

The three examples provided are for USB device mode only, but the architecture can be followed for USB host mode as well. The examples cover the CDC, Bulk, and HID modes that are the most frequently used USB modes. Adding support for additional USB modes that are part of the TivaWare USB Library can be done by referencing the corresponding TivaWare projects.

4.2.1 usb_dev_bulk

This example configures the USB peripheral to function as a USB device in bulk mode. In order to observe USB bulk transfers, an additional program is needed which communicates with USB bulk devices. A USB bulk example program is provided as part of TivaWare and can be downloaded from the SW-TM4C tool page. The software package to install the USB bulk example program is titled *SW-USB-win-2.2.0.295.msi*. After installing the program, navigate to the installation folder and find *usb_bulk_example.exe*. On opening, the application looks to connect to a TM4C USB device, so have the usb_dev_bulk example project loaded and running before starting the demo application.

In this example, only one task is created to handle the USB bulk data transfers. A counting semaphore is used to allow the USB handler to indicate when data has been received through the bulk interface. Once data is received, the task is unblocked and it will read out the received data and then echo it back over USB bulk while inverting the casing of any letters. When used with the *usb_bulk_examples.exe* program, data sent will be displayed in 255 byte packets as that is the maximum size of individual USB bulk packets under the USB 2.0 Full Speed standard. Messages larger than 255 bytes can be sent, and will be received across multiple returned packets.

4.2.2 usb_dev_cdcserial

This example configures the USB peripheral to function as a USB device in CDC mode and configures the UART0 peripheral to be able to both transmit and receive data. The example demonstrates a UART to USB bridge with bi-directional communication. To run the full demonstration, two USB cables are needed. One will connect to the Stellaris Virtual COM Port which runs through UART0 and the other will connect to the USB bus of the TM4C microcontroller which will enumerate as a serial COM port when used as a CDC device. Using COM port terminal software, messages can be sent back and forth between UART and USB.

In this example, three tasks are created to manage the UART to USB bridge. The UART to USB task blocks a task notification is given from the UART RX interrupt. Once unblocked, the task checks for how much USB TX buffer space is available before reading in bytes of UART data to write back over USB. It also gathers data on any UART errors received and if any errors occur that information is sent to the error handler via a queue.

The USB to UART task blocks until a task notification is given from the USB RX handler to indicate data has been received. It will read out the USB data from the USB RX buffer and send the bytes of UART. To ensure the UART TX FIFO has enough time to send messages, TX interrupts are enabled and monitored as part of this process. Furthermore, a mutex is used to guard access to the task as USB messages can be received and processed faster than UART transfers can be completed.

The final task is an error handler that waits for error data to arrive via a queue, and if received it processes what errors occurred and reports them back to the USB stack so the CDC driver is aware of the errors that occurred.

4.2.3 usb_dev_keyboard

This example showcases how to configure the USB peripheral to function in USB HID device mode while enumerating the HID device as a USB keyboard peripheral. By enumerating as a keyboard, it is possible for the microcontroller to type out text on the screen exactly like a user writing on a keyboard. The messages that are typed out are controlled by the push buttons on the LaunchPad. One button will always output a default greeting message. The other can print out a second default message that can be updated by sending a new message to the Stellaris Virtual COM Port with a COM port terminal. The message must be appended with a CR-LF to be recognized as completed. The message size is limited to 255 bytes by default and sending a message longer than 255 bytes will result in the message being broken up. The default size can be increased in the project by editing the *UART_BUFFER_SIZE* variable. If three or more messages are sent sequentially without printing any messages out, only the first two will be stored for display.

In this example, two tasks are created. The first task manages updating the dynamic message by receiving data over UART and sending completed messages to the USB output task through a queue. The messages are checked to see if they exceed the maximum size or if the last two bytes received correspond to a Carriage Return-Line Feed (CR-LF) combination. Once one of those conditions is cleared, the queue is used to pass the updated message information to the USB output task.

The second task that handles the output of the USB keyboard message waits in a blocked state until a semaphore is given based on the push buttons. Pressing a button will trigger an ISR for the button that gives the output task the semaphore. The output task then checks which button was pressed and sends the appropriate message. If the button press is for the message that can be updated, the output task first checks if it has received a new message in the queue from the UART peripheral.

4.3 Ethernet Examples

Most TM4C129x microcontrollers come with an Ethernet peripheral that includes integrated Ethernet hardware components. Depending on the specific device, the Ethernet peripheral can include an integrated Ethernet MAC with MII/RMII connections to support an external Ethernet PHY, having both the Ethernet MAC and PHY integrated into the microcontroller with signal access to the PHY only, or having both the MAC and PHY integrated with the signal access options for both the PHY and MI/RMII interfacing.

The EK-TM4C1294XL LaunchPad features a TM4C microcontroller that has the integrated Ethernet MAC+PHY combination and the board comes with an Ethernet jack to connect to the peripheral. Two examples are provided for this LaunchPad to demonstrate how to use the lightweight IP (lwIP) stack with FreeRTOS to create Ethernet-enabled applications.

4.3.1 enet_lwip

This example application demonstrates the operation of the TM4C Ethernet controller using the lwIP TCP/IP Stack. In this example the EK-TM4C1294XL LaunchPad acts as a webserver. As a webserver, the EK-TM4C1294XL LaunchPad receives incoming network HTTP requests and sends outgoing HTTP responses along with web contents through TCP/IP connections. 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 instead. The selected address is then output over the UART interface.

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.

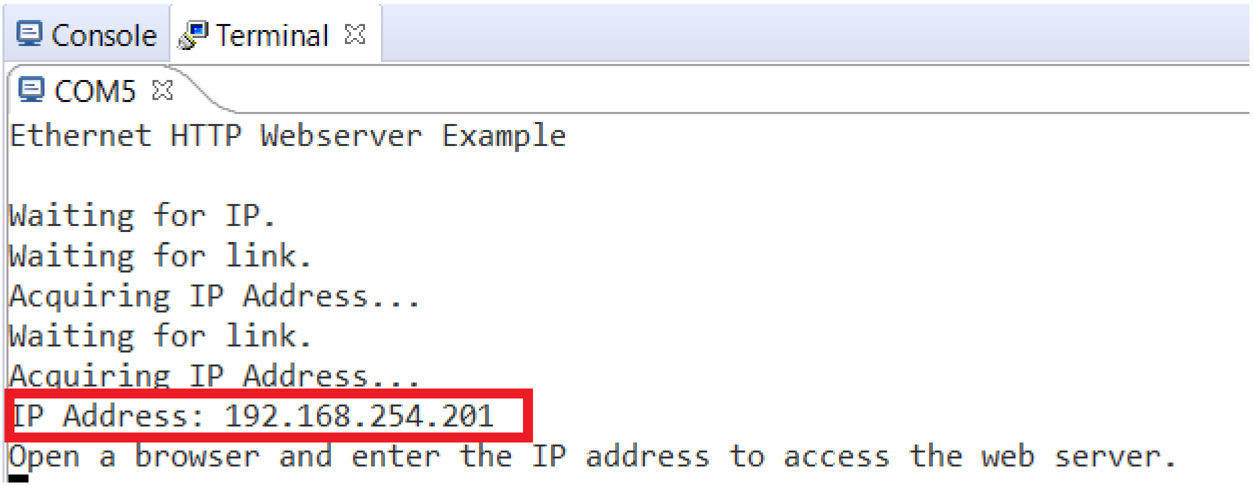
```
../../../../tools/bin/makefsfile -i fs -o enet_fsdata.h -r -h -q
```

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

4.3.1.1 Running enet_lwip Example

After the application firmware is loaded and run, the example will display various information such as the acquired DHCP IP address and the progress of the communication on the terminal emulator. Any terminal emulator that connects to a serial COM port may be used. The terminal emulator should be configured for 115200 Baud and 8-N-1.

As can be seen in figure Figure 4-5, the IP address that is acquired is 192.168.254.201 when this example is run on a given network . A different IP address will be assigned when the same example is run on a different network.

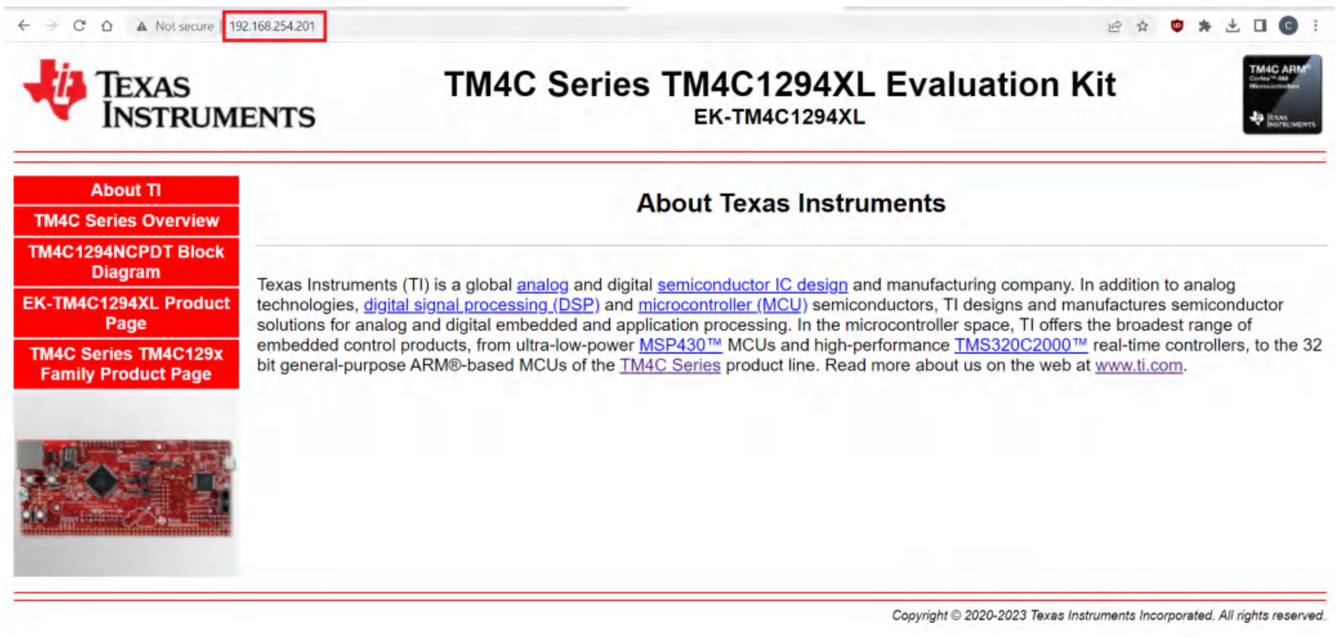


```
Ethernet HTTP Webserver Example

Waiting for IP.
Waiting for link.
Acquiring IP Address...
Waiting for link.
Acquiring IP Address...
IP Address: 192.168.254.201
Open a browser and enter the IP address to access the web server.
```

Figure 4-5. DHCP IP Address

Enter the IP address that is shown in the terminal window to the URL field of the browser. The webserver displays the web contents on the browser. Click the various menu items to explore the different web contents.



TM4C Series TM4C1294XL Evaluation Kit

EK-TM4C1294XL

About TI
TM4C Series Overview
TM4C1294NCPDT Block Diagram
EK-TM4C1294XL Product Page
TM4C Series TM4C129x Family Product Page

About Texas Instruments

Texas Instruments (TI) is a global analog and digital semiconductor IC design and manufacturing company. In addition to analog technologies, digital signal processing (DSP) and microcontroller (MCU) semiconductors, TI designs and manufactures semiconductor solutions for analog and digital embedded and application processing. In the microcontroller space, TI offers the broadest range of embedded control products, from ultra-low-power MSP430™ MCUs and high-performance TMS320C2000™ real-time controllers, to the 32 bit general-purpose ARM®-based MCUs of the TM4C Series product line. Read more about us on the web at www.ti.com.

Copyright © 2020-2023 Texas Instruments Incorporated. All rights reserved.

Figure 4-6. HTTP Webserver Main Page

4.3.2 enet_io

This example application demonstrates web-based I/O control using the TM4C 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 the user to access the internal web pages served by the application via a 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 the user to toggle the state of the LED on the board and set the rate for blinking the 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 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.

The web server for this example has been modified from the example offered in 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.

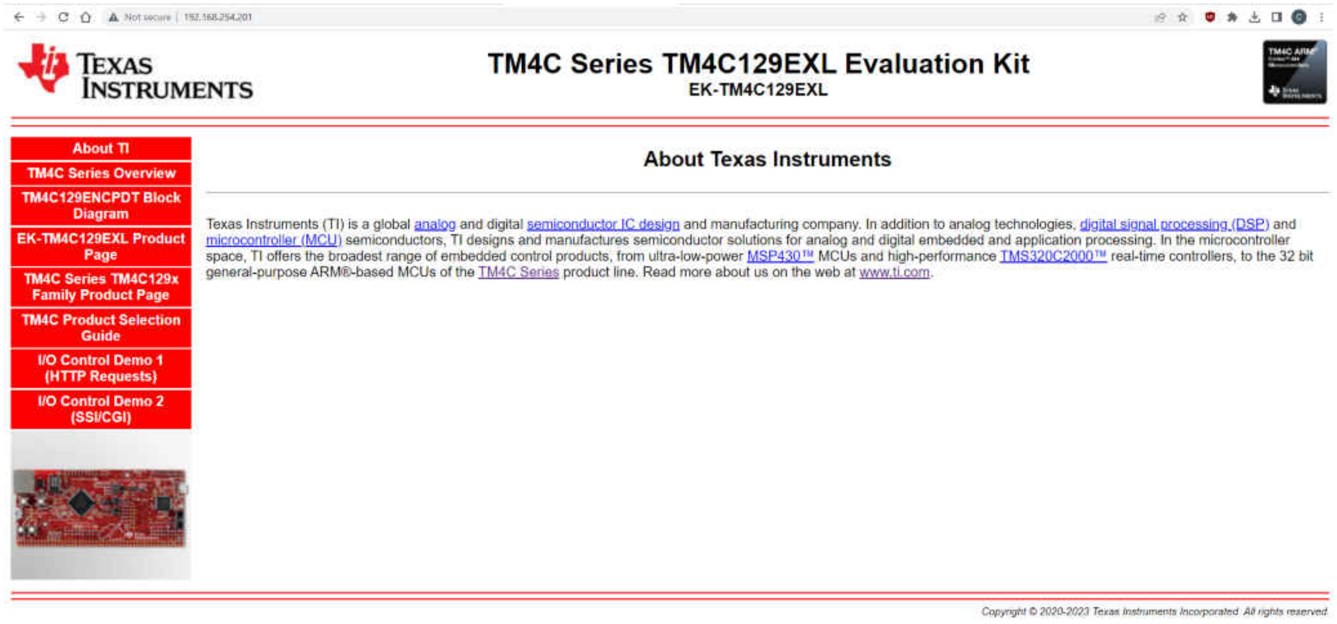
Source files for the internal file system image can be found in the `fs` directory. If any of these files are changed, the file system image, `io_fsdata.h`, must be rebuilt by running the following command from the `enet_io` directory:

../../../../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/

4.3.2.1 Running enet_io Example

Similar to the enet_lwip example, this example will display the acquired DHCP IP address on the terminal window. Enter the IP address to the URL of the browser. The web page displays web contents similar to enet_lwip but with three additional menu items. Click the IO Control Demo 1 and the IO Control Demo 2 menu items to control the on-board LED and the animation speed.



Not secure | 192.168.254.201

TM4C Series TM4C129EXL Evaluation Kit
EK-TM4C129EXL

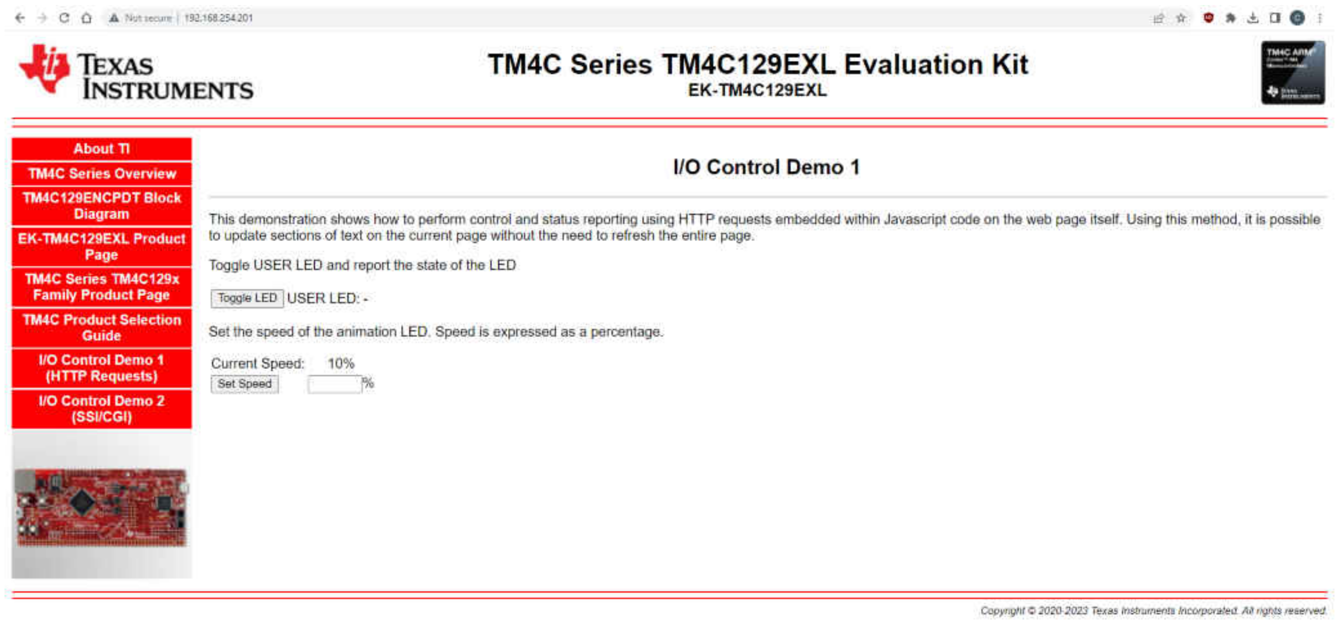
About TI
TM4C Series Overview
TM4C129ENCPDT Block Diagram
EK-TM4C129EXL Product Page
TM4C Series TM4C129x Family Product Page
TM4C Product Selection Guide
I/O Control Demo 1 (HTTP Requests)
I/O Control Demo 2 (SSI/CGI)

About Texas Instruments

Texas Instruments (TI) is a global analog and digital semiconductor IC design and manufacturing company. In addition to analog technologies, digital signal processing (DSP) and microcontroller (MCU) semiconductors, TI designs and manufactures semiconductor solutions for analog and digital embedded and application processing. In the microcontroller space, TI offers the broadest range of embedded control products, from ultra-low-power MSP430™ MCUs and high-performance TMS320C2000™ real-time controllers, to the 32 bit general-purpose ARM®-based MCUs of the TM4C Series product line. Read more about us on the web at www.ti.com.

Copyright © 2020-2023 Texas Instruments Incorporated. All rights reserved.

Figure 4-7. Ethernet IO Main Page


Not secure | 192.168.254.201
TM4C Series TM4C129EXL Evaluation Kit
EK-TM4C129EXL
About TI
TM4C Series Overview
TM4C129ENCPDT Block Diagram
EK-TM4C129EXL Product Page
TM4C Series TM4C129x Family Product Page
TM4C Product Selection Guide
I/O Control Demo 1 (HTTP Requests)
I/O Control Demo 2 (SSI/CGI)
Toggle LED
Set Speed
Copyright © 2020-2023 Texas Instruments Incorporated. All rights reserved.

I/O Control Demo 1

This demonstration shows how to perform control and status reporting using HTTP requests embedded within Javascript code on the web page itself. Using this method, it is possible to update sections of text on the current page without the need to refresh the entire page.

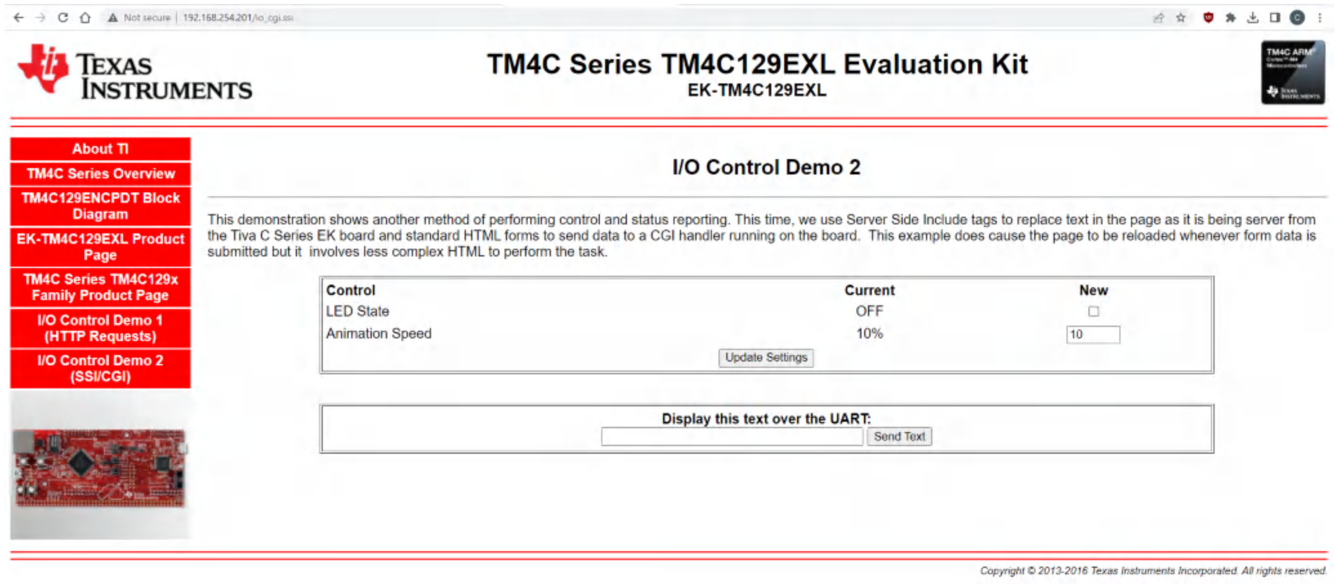
Toggle USER LED and report the state of the LED

USER LED: -

Set the speed of the animation LED. Speed is expressed as a percentage.

Current Speed: 10%

Figure 4-8. IO Control Demo 1 Webpage Using HTTP Requests


Not secure | 192.168.254.201/io_cgi.ssi
TM4C Series TM4C129EXL Evaluation Kit
EK-TM4C129EXL
About TI
TM4C Series Overview
TM4C129ENCPDT Block Diagram
EK-TM4C129EXL Product Page
TM4C Series TM4C129x Family Product Page
I/O Control Demo 1 (HTTP Requests)
I/O Control Demo 2 (SSI/CGI)
Update Settings
Send Text
Copyright © 2013-2016 Texas Instruments Incorporated. All rights reserved.

I/O Control Demo 2

This demonstration shows another method of performing control and status reporting. This time, we use Server Side Include tags to replace text in the page as it is being server from the Tiva C Series EK board and standard HTML forms to send data to a CGI handler running on the board. This example does cause the page to be reloaded whenever form data is submitted but it involves less complex HTML to perform the task.

Control	Current	New
LED State	OFF	☐
Animation Speed	10%	10

Display this text over the UART:

Figure 4-9. IO Control Demo 2 Webpage Using SSI/CGI

5 References

- Texas Instruments: *Developing Basic Applications With FreeRTOS on TM4C MCU*
- Texas Instruments: *Developing Common Applications With FreeRTOS on TM4C MCUs*
- Texas Instruments: *Getting Started with TivaWare™ User's Guide*
- Texas Instruments: *Using USB Host Mode on the EK-TM4C123GXL LaunchPad*