Application Report

TM4C129x Ethernet Applications for Socket-Based TI-RTOS NDK

Charles Tsai

ABSTRACT

The TM4C129x MCU is a highly-connected 32-bit Arm® Cortex®-M4F microcontroller with integrated Ethernet MAC and PHY and a wide variety of wire communication peripherals. This application report demonstrates various Ethernet application examples based on the TI-RTOS NDK – a platform for development of network enabled applications on TI embedded processors. The EK-TM4C1294XL LaunchPad™ Development Kit for which the examples are run on is an evaluation platform for the TM4C129x Ethernet MCUs.

Project collateral and source code discussed in this document can be downloaded from the following URL: https://www.ti.com/lit/zip/spma080.

Table of Contents

List of Figures

List of Tables

Trademarks

1 Introduction

The Network Developer's Kit (NDK) is a platform for development and demonstration of network enabled applications on TI embedded processors. The code included in the NDK is generic C code which runs on a variety of TI devices.

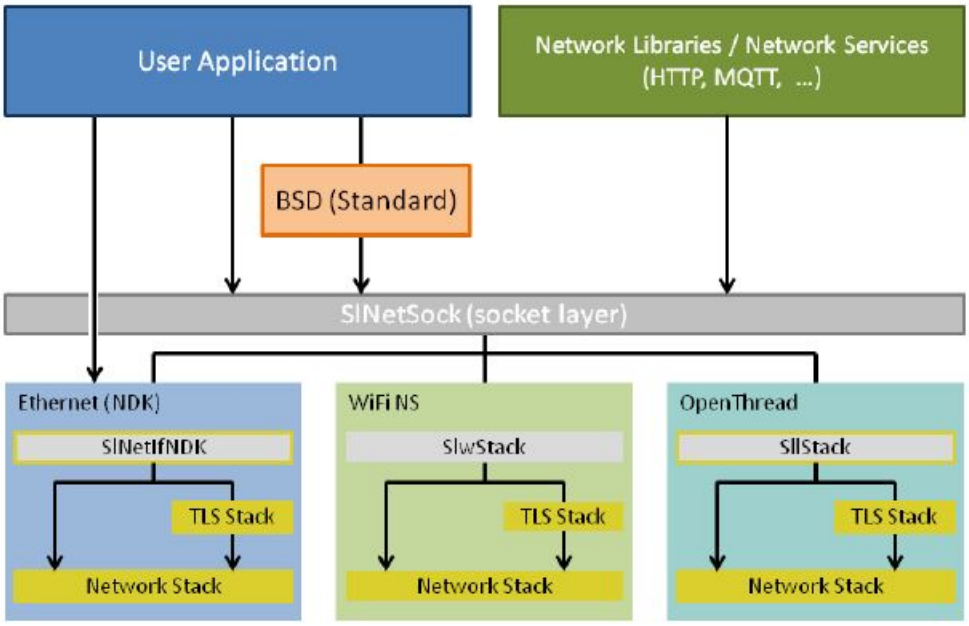


Figure 1-1. Components of a NDK-Enabled Application

In Figure 1-1, the user application can make calls using the standard BSD sockets APIs, or it can directly call into the SlNetSock layer to communicate with a network connection. The SlNetSock layer is a stack-independent layer between the user application and the service-specific stacks.

The NDK is designed to provide a platform-independent, device-independent, and RTOS-independent API interface for use by the application.

NDK is a networking component of TI-RTOS - a scalable embedded tools ecosystem for TI devices. TI-RTOS scales from a real-time multitasking kernel (SYS/BIOS) to a complete RTOS solution with middleware components and device drivers.

The main focus of this application report is to illustrate the various Ethernet applications for TI-RTOS NDK. This application report assumes the user has some basic understanding about TI-RTOS and NDK. If you are new to TI-RTOS and NDK, refer to the below documents for details:

- *TI Network Developer's Kit (NDK) User's Guide*
- *TI Network Developer's Kit (NDK) API Reference Guide*
- TI-RTOS-MCU Overview – Product Page
- *TI-RTOS for TivaC Getting Started Guide*
- *TI-RTOS User's Guide*
- *SYS/BIOS (TI-RTOS Kernel) User's Guide*

1.1 TI-RTOS Download

Download TI-RTOS for Tiva C (TM4C) from: http://downloads.ti.com/dsps/dsps_public_sw/sdo_sb/targetcontent/tirtos/index.html.

1.2 License

TI-RTOS is under the BSD license. When downloading TI-RTOS, see the License Agreement.

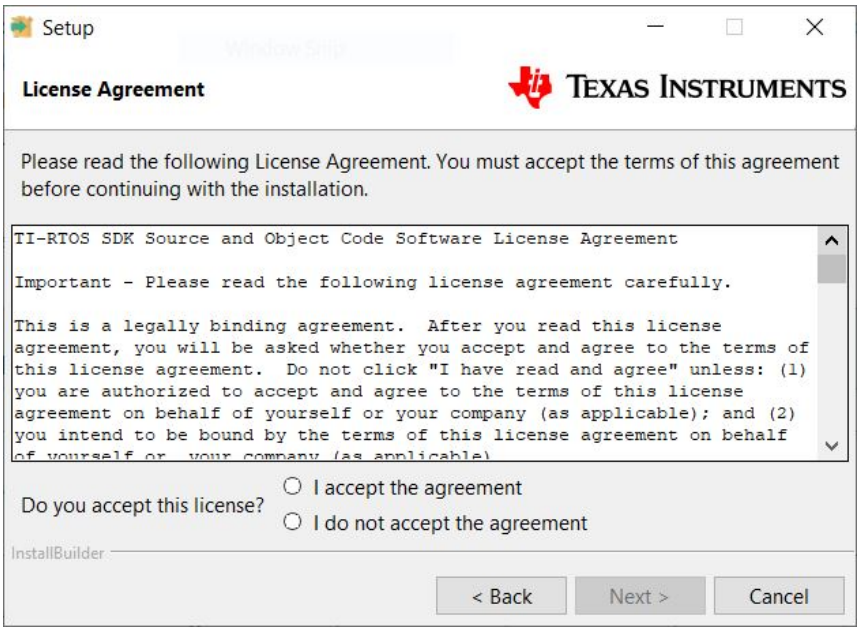


Figure 1-2. TI-RTOS License Agreement

1.3 XDCtools

XDCtools is a separate software component provided by Texas Instruments that provides the underlying tooling needed for configuring and building SYS/BIOS, NDK and UIA. Normally, the XDCtools is automatically installed as part of the Code Composer Studio™ (CCS) installation.

Part of TI-RTOS installation is to install XDCtools only if the versions needed by TI-RTOS has not already been installed as part of a CCS or SYS/BIOS installation.

1.4 Versions

TI-RTOS version 2.16.xx.xx which contains NDK version 2.25.00.09 is the latest version for Tiva C (TM4C) development.

The recommended XDCtools version for Tiva C (TM4C) is v3.32.0.06.

Note

Do not use any newer TI-RTOS versions than 2.16.xx.xx for Tiva C (TM4C). Do not use any newer XDCtools versions than 3.32.0.06.

1.5 Configuring NDK Modules

To configure the NDK and its components, the NDK allows you to use either C code to call functions or the XGCONF configuration tool within CCStudio.

- C Code: Configure the application by writing C code that calls *CfgNew()* to create a configuration database and other *Cfg*()* functions to add various settings to that configuration database. For details, refer to section 2.1 of SPRU523. This configuration method is recommended because it can be used with any RTOS supported by the SDK.
- XGCONF: The XGCONF is a GUI within CCStudio to enable and set properties. When you create a project using the TI-RTOS Kernel template, the project will contain a configuration file (*.cfg) that can be edited with the XGCONF graphical editor in CCSstudio. The configuration file is processed during the build to generate code that configures your application. This configuration method can be used only if your RTOS is the TI-RTOS Kernel.

All examples presented in this application use XGCONF to configure the NDK modules.

1.6 Socket-Based API

The application examples provided in this application report use the standard BSD APIs for socket actions such as accept, send, and receive. As shown in Figure 1-1, the standard BSD APIs are provided via SINetSock. It is also possible to use the pure NDK style socket API functions. The NDK socket APIs will have the prefix of NDK_*() as in NDK_accept(), NDK_send() and NDK_receive(). While the NDK_*() socket calls use a little smaller footprint and have slightly better performance, the difference is small.

1.6.1 BSD Style Socket APIs

Table 1-1 lists a typical BSD socket APIs. Figure 1-3 shows a simplified flowchart of a TCP client-server communication using the BSD socket APIs. Figure 1-4 shows the APIs usage of a UDP communication.

Table 1-1. BSD Socket APIs

Function Category	API	Description
Socket connection	socket	Creates a new socket of a certain type, identified by an integer number, and allocates system resources to it.
	bind	Is typically used on the server side, and associates a socket with a socket address structure, i.e. a specified local IP address and a port number.
	listen	Is used on the server side, and causes a bound TCP socket to enter listening state.
	accept	Is used on the server side. It accepts a received incoming attempt to create a new TCP connection from the remote client, and creates a new socket associated with the socket address pair of this connection.
	connect	Is used on the client side, and assigns a free local port number to a socket. In case of a TCP socket, it causes an attempt to establish a new TCP connection.
Receiving data	recv	Used for receiving data. Normally used only on a connected socket.
	recvfrom	Used for receiving data. May be used to receive data on a socket whether or not it is connection-oriented.
Sending data	send	Used for sending data. Normally used for TCP SOCK_STREAM connected sockets.
	sendto	Used for sending data. Normally used for UDP SOCK_DGRAM unconnected datagram sockets.

Table 1-1. BSD Socket APIs (continued)

Function Category	API	Description
I/O multiplexing	poll	Is used to check on the state of a socket in a set of sockets. The set can be tested to see if any socket can be written to, read from or if an error occurred.
	select	Is used to suspend, waiting for one or more of a provided list of sockets to be ready to read, ready to write, or that have errors.
Closing connection	close	Close the TCP connection.
Socket options	setsocketopt	Is used to set a particular socket option for the specified socket.
	getsockopt	Is used to retrieve the current value of a particular socket option for the specified socket.

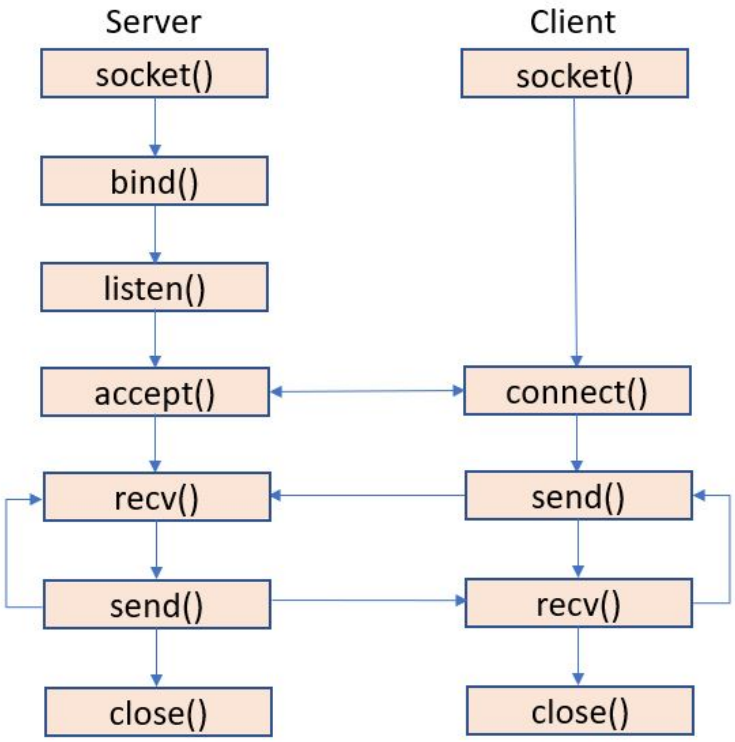


Figure 1-3. TCP Client Server Communication using BSD Socket APIs

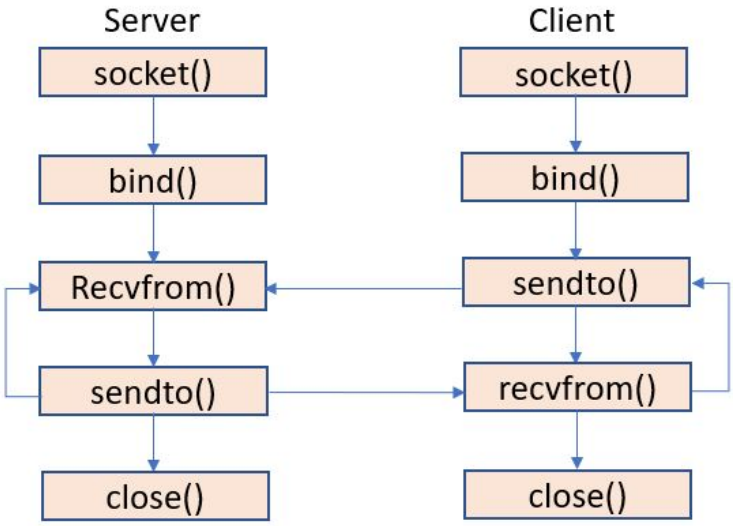


Figure 1-4. UDP Client Server Communication using BSD Socket APIs

2 Application Examples

A collection of eight examples are presented here to show the TM4C129x MCU running either the Ethernet server applications or client applications using TI-RTOS NDK.

Table 2-1. Application Examples

Example	Type	Description
enet_tcpecho_server_tirtos	Server	An echo server application using TCP protocol. The server echoes back the packets received from the client. Dynamic IP address is acquired from a DHCP server.
enet_updecho_server_tirtos	Server	An echo server application using UDP protocol. The server echoes back the datagrams it receives from the client.
enet_httpServer_tirtos	Server	A HTTP web server application that is hosting web pages.
enet_dns_tirtos	Client	A client application that requests the DNS (Domain Name Server) server to translate domains names into IP addresses, making it possible for the DNS clients to reach the origin server.
enet_sntp_tirtos	Client	A client application that reports the current network time based on the SNTP (Simple Network Time Protocol).
enet_tcpecho_client_tirtos	Client	An echo client application using TCP protocol. The client sends a greeting message to the server and echoes back the packets it receives from the server.
enet_udpecho_client_tirtos	Client	An echo client application using UDP protocol. The client sends a greeting message to the server and echoes back the datagrams it receives from the server.
enet_httpget_tirtos	Client	A client application that sends a HTTP GET request to an internet server.

3 Application Setup

3.1 Hardware Setup

- A network router that connects to the internet
- An Ethernet switch that connects devices to the LAN network. The switch is optional if you can connect the devices to the router directly. Most of the home routers have the LAN ports for wired connections to devices.
- A PC is used to:
  - Debug the target device.
  - Act as either a server or client to generate or respond to network traffic to/from the target device.
  - Monitor the network traffic.
- The EK-TM4C1294XL LaunchPad evaluation kit that runs the applications provided in this application report.

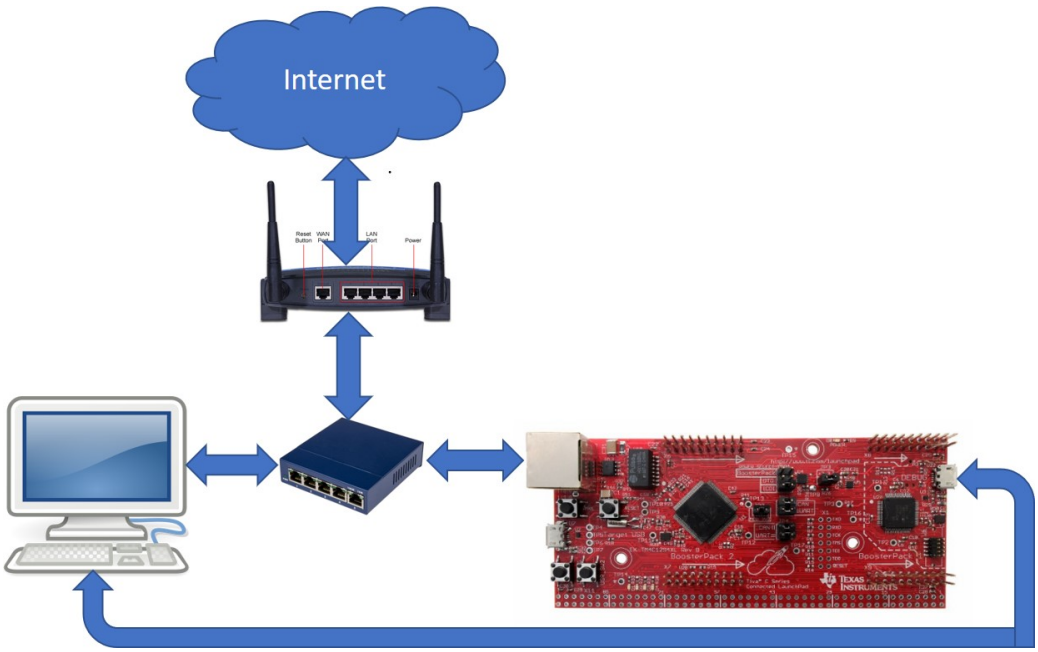


Figure 3-1. Hardware Setup for the Application Examples

3.2 Software Tools

Several tools are used to facilitate the debugging and testing of the application examples:

- CCS. This IDE tool is used to debug the target device and build the application examples. CCS v10.1.1 and TI compiler v20.2.4.LTS are used for this application report.
- SocketTest. A free small software tool that tests any server or client that uses TCP or UDP protocol to communicate.
- Wireshark. A free packet analyzer that is used for network troubleshooting and analysis in this application report.

4 Download and Import the Ethernet Examples

There are eight CCS project examples attached as collateral to this document. You can unzip the project or keep it in the zip format. Both formats can be imported by CCS.

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

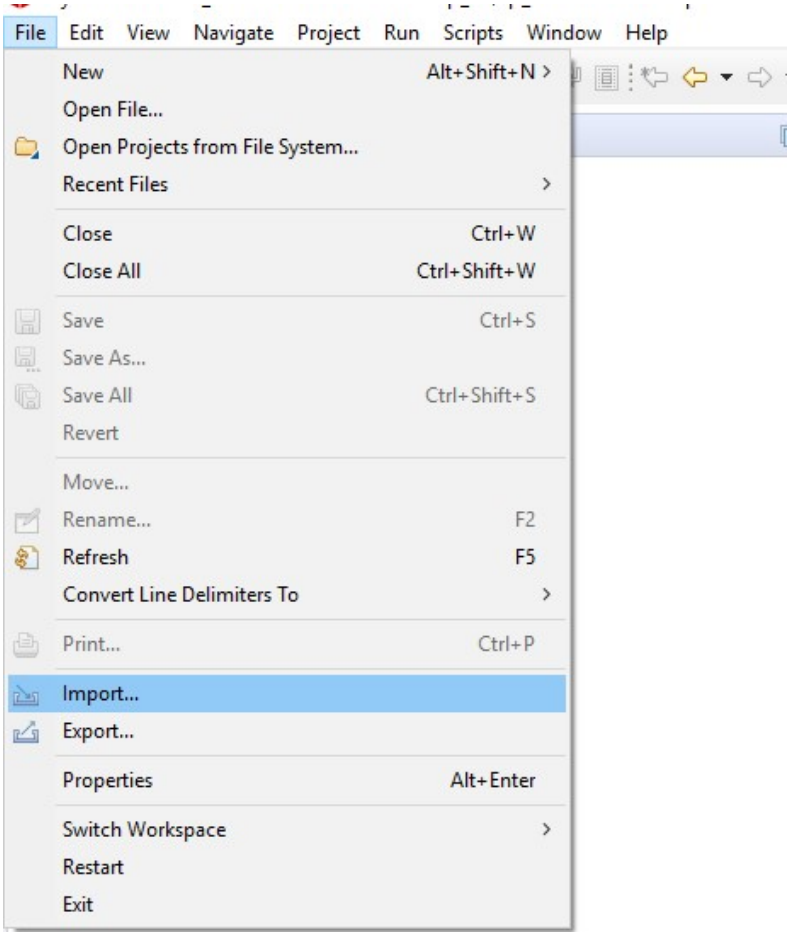


Figure 4-1. Import CCS Projects Step 1

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

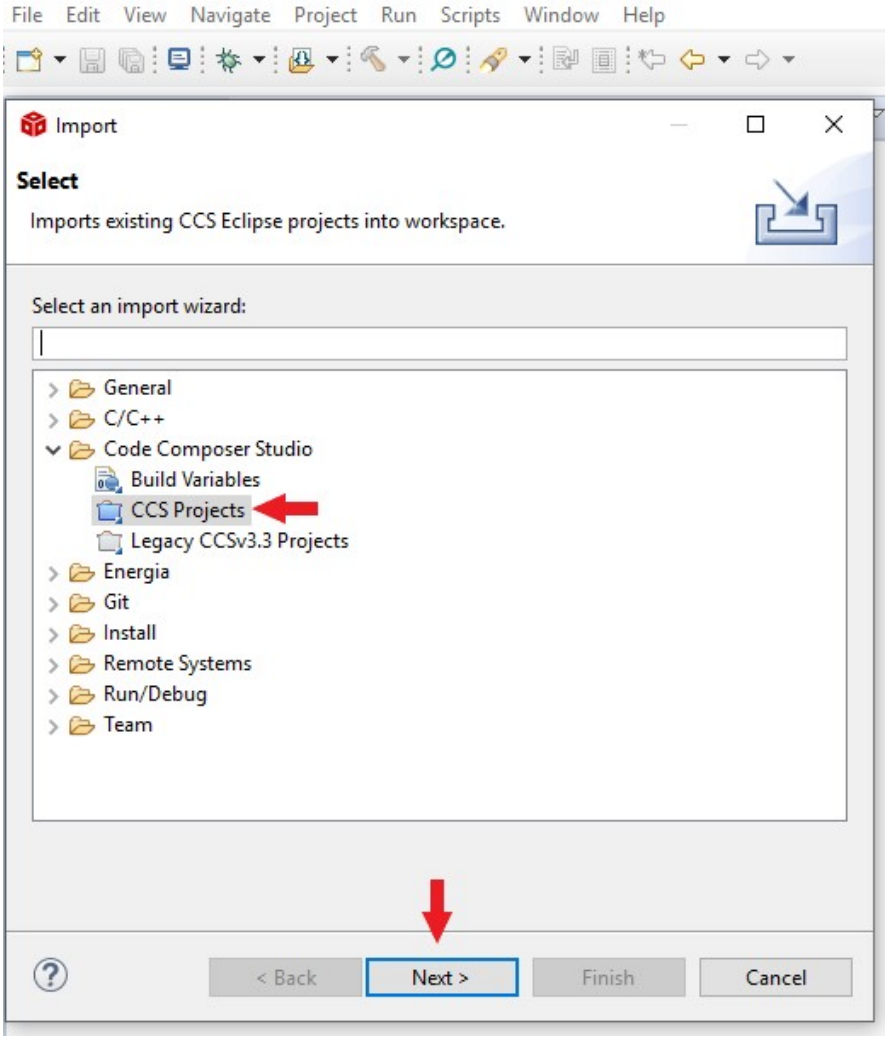


Figure 4-2. Import CCS Projects Step 2

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”.

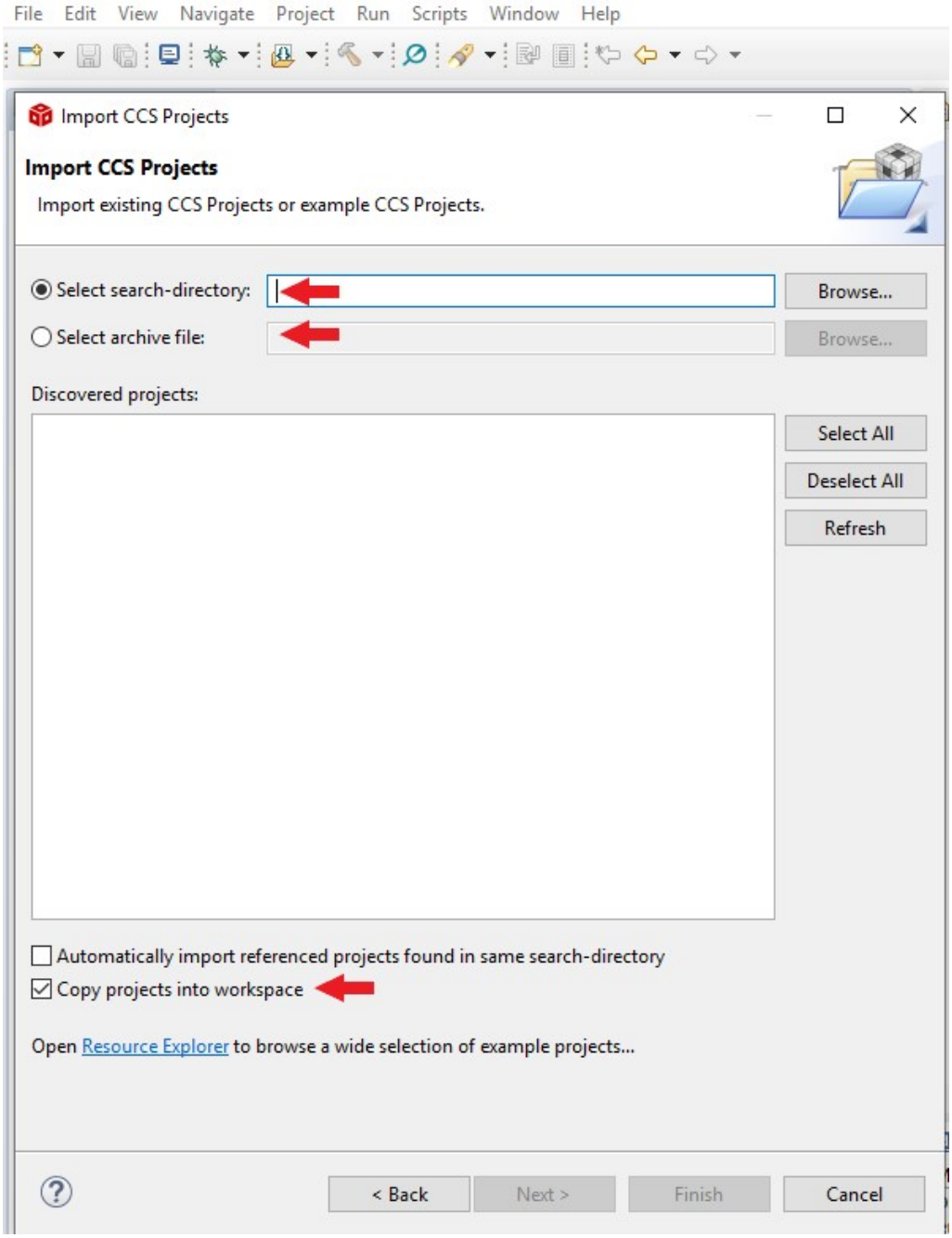


Figure 4-3. Import CCS Projects Step 3

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

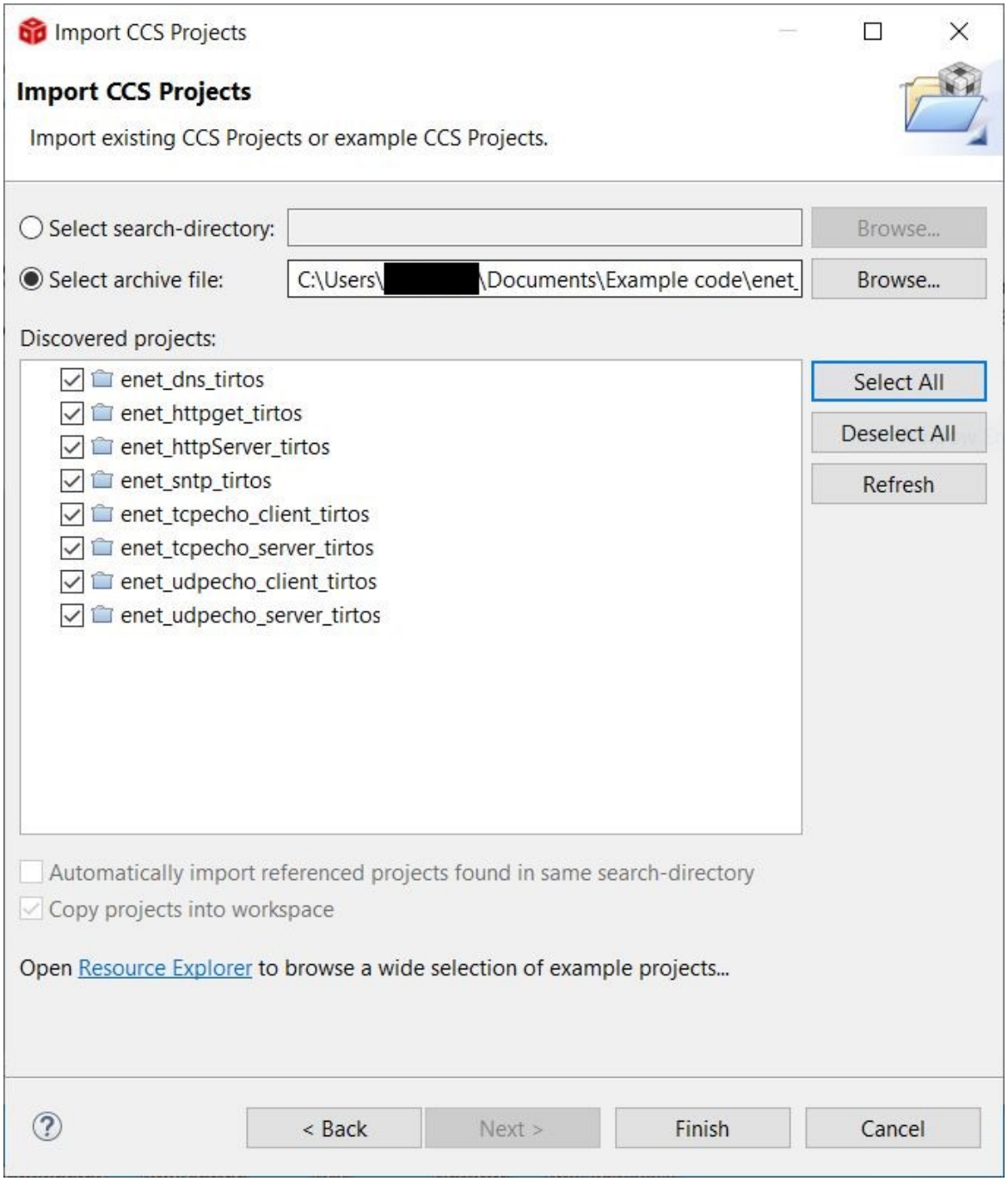


Figure 4-4. Import CCS Projects Step 4

5 How to Create an Ethernet Application for TI-RTOS NDK

The best way to start developing a TI-RTOS NDK application is to reference an existing example. Follow the steps below to view the NDK configuration for the enet_tcpecho_server_tirtos example.

1. Every TI-RTOS project will contain a *.cfg file if the XGCONF is the method used for configuration as explained in Section 1.5. The *.cfg is used to add and configure various TI-RTOS and NDK modules and parameters. First, right-click on the *.cfg file, see Figure 5-1.
2. Select “XGCONF” under the “Open With” submenu. Once done, you will first see a TI-RTOS Welcome page.
3. Click the “Global” in the “Outline” tab will lead you to the NDK Welcome page. Next click the “System Overview” tab shown in box 5.
4. Various NDK modules will be displayed graphically as shown in Figure 5-1. For example, you will see TCP, UDP and NAT modules in the Transport Layer. Depending on which modules/features you will need for your application, you will click the modules to enable them. For this example, only the TCP, UDP and the IP modules are enabled. Other examples in this application report may require different NDK modules to be enabled. Once enabled, you will see a green check mark. For instance, clicking the Internet Protocol (IP) module will take you to IP module configuration shown in Figure 5-2. Here you can enable the module and choose to obtain the dynamic IP address automatically or you can provide the static IP address.
5. In box 5, various global NDK settings are partitioned into different tabs. Click on each one of them to get a feel for what they are. See Figure 5-3 where “netOpenHook” is defined as a user supplied hook function. The Network open hook function is called when the stack is ready to begin the creation of application supplied network tasks. You can see the definition of this function in the tcpEchoHooks.c file of the project. Other examples may use different hook functions. For instance, the enet_tcpecho_client_tirtos example will plug the hook function for Network IP address hook because the client will wait until it obtains the IP address before it attempts to connect to the server.
6. You can also directly edit the Javascript-like *.cfg file for both the TI-RTOS and NDK configurations.

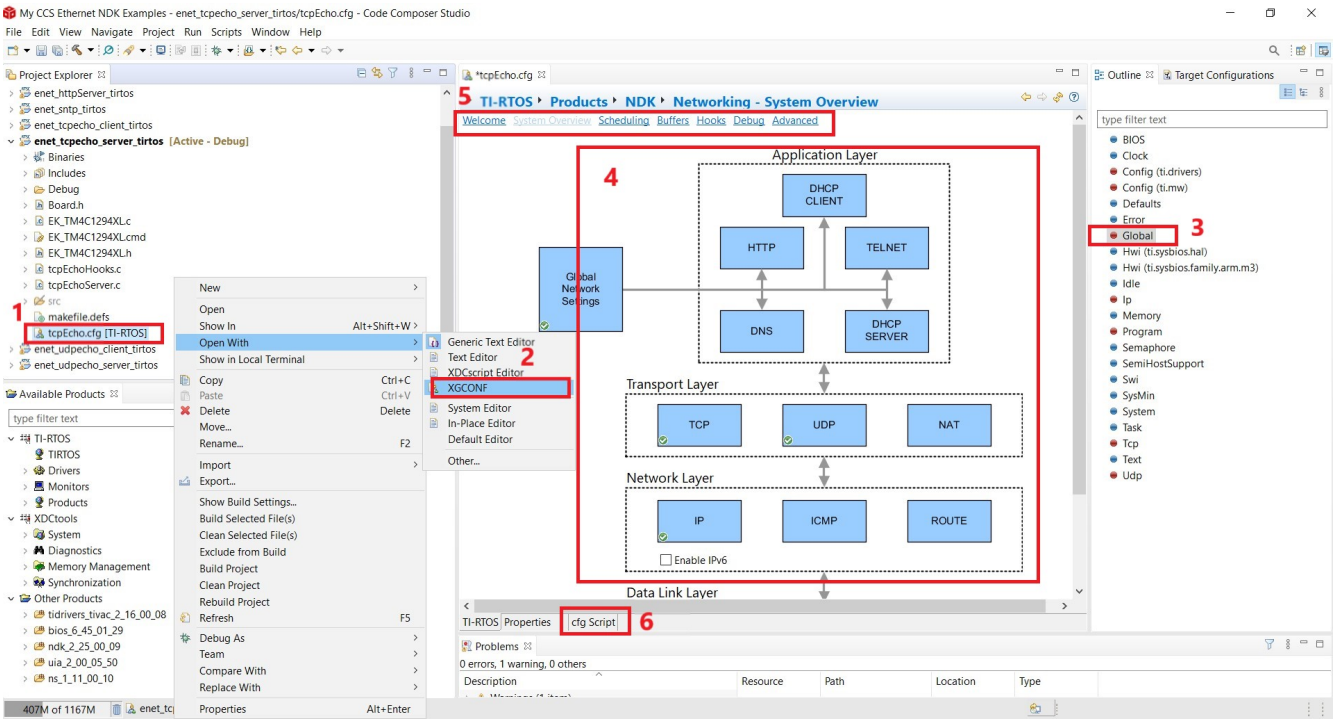


Figure 5-1. NDK Configuration Using XGCONF

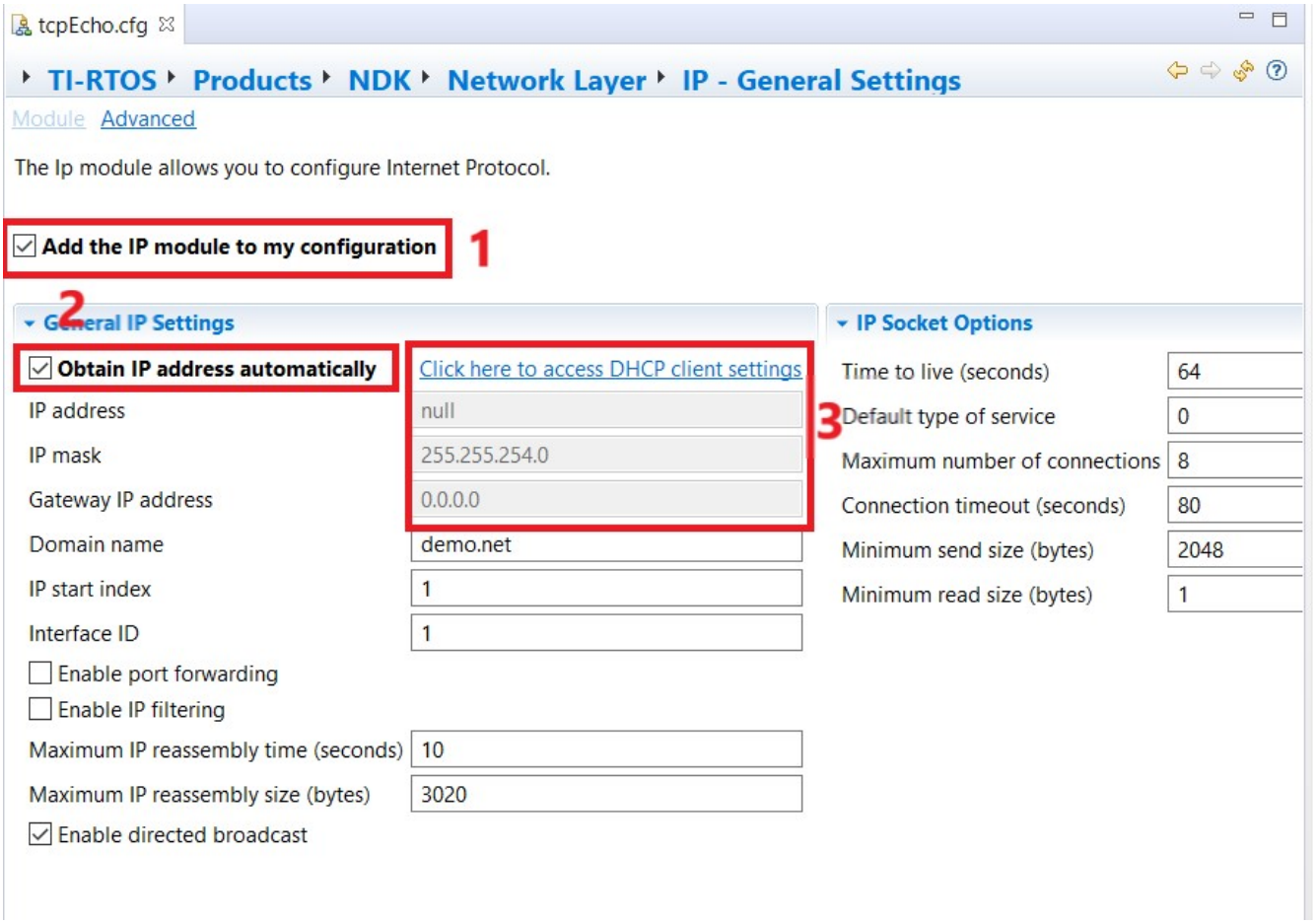


Figure 5-2. IP Module Configuration

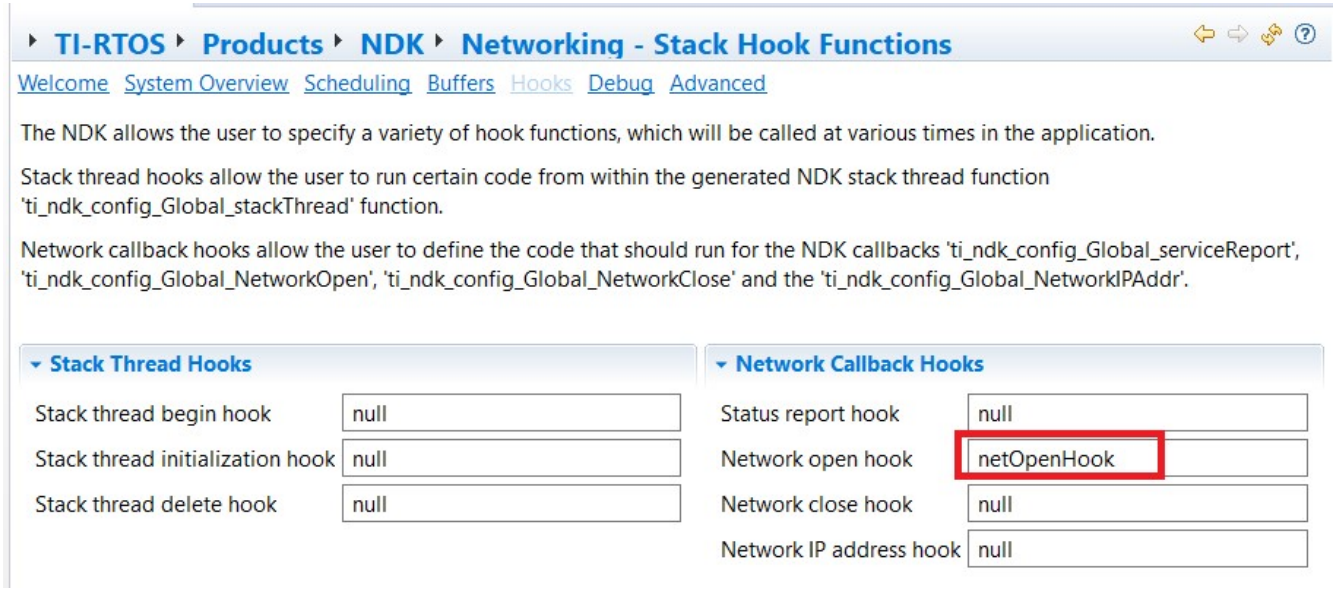


Figure 5-3. Hooks Function Configuration

6 Enet_tcpecho_server_tirtos Example Overview

The enet_tcpecho_server_tirtos example demonstrates an echo-server application running on the TM4C129x MCU using Transmission Control Protocol (TCP) as the underlying transport layer protocol. TCP is a connection-oriented protocol with built in error recovery and retransmission. The connection protocol is likened to a telephone connection. Both the sender and the receiver need to handshake for the connection (for example, the caller calls the number and the callee picks up the call) before communicating. The connection is there until one party hangs up the connection. TCP is used by applications when guaranteed message and error-free delivery is required.

In this example, the TM4C129x MCU is acting as a server. The NDK stack is configured for DHCP to automatically acquire an IP address. Once acquired, the IP address is displayed on the CCS console window. The echo-server is ready by this time. The server will listen for the connection from the client. Once the client makes a connection the communication between the server and the client can start. The server implemented in this example will read the received characters and then echo the characters back to the client.

6.1 Build and Flash the Program

First select the enet_tcpecho_server_tirtos as the active project. With the LaunchPad's ICDI USB port connected to the PC via the USB cable, build the project and load the program by clicking the Debug icon.

Click on the CCS User's Guide if you are new to CCS.

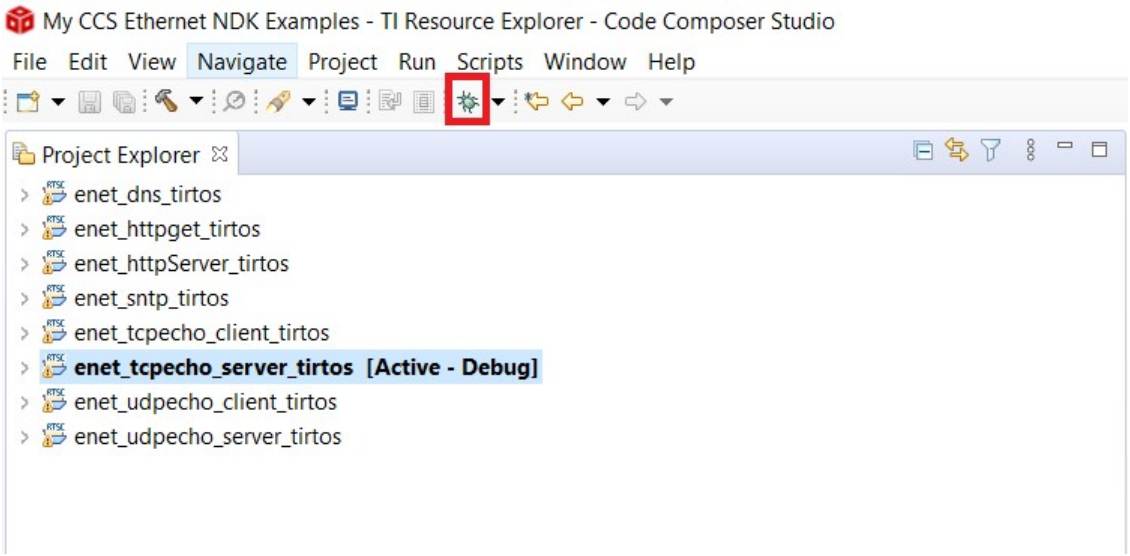


Figure 6-1. Debug CCS Projects

6.2 Check and Program the MAC Address

Every NIC (Network Interface Controller) on the network must be uniquely identified by a MAC address for communications within a network segment. A MAC address is a 48-bit value represented as six octets of two hexadecimal digits. MAC addresses are primarily assigned by the device manufactures. The first three octets are the OUI (Organizationally Unique Identifier). The MAC address is normally pre-programmed on the EK-TM4C1294XL LaunchPad board. There is also a sticker on the back of the LaunchPad with the written MAC address. The pre-programmed MAC addresses will have the first three octets equal to 00:1A:B6 which uniquely identifies Texas Instruments. If you have a virgin device then the MAC address is not pre-programmed. You must program the MAC address yourself with the addresses that are allocated to your organization.

There are three tools to read and program the MAC address.

6.2.1 Using The LM Flash Programmer

This tool is most suitable if you are using the EK-TM4C1294XL LaunchPad. The LM Flash Programmer only supports the ICDI debug probe which is built-in on the LaunchPad. To check and program the MAC address follows the below steps.

1. Open the LM Flash Programmer and go to the “Other Utilities” tab.
2. Select the “MAC Address Mode” radio button.
3. To read the MAC address, press the “Get Current MAC Address” button.
4. To program the MAC address:
   a. first type the six-octet MAC address into the “MAC Address” field.
   b. Next, click the “Commit MAC Address” checkbox. This commit will permanently store the MAC address on the internal EEprom. If the commit is not checked, the MAC address just entered will be temporary until the next power cycle.
   c. Finally, press the “Program MAC Address” button to complete the programming.

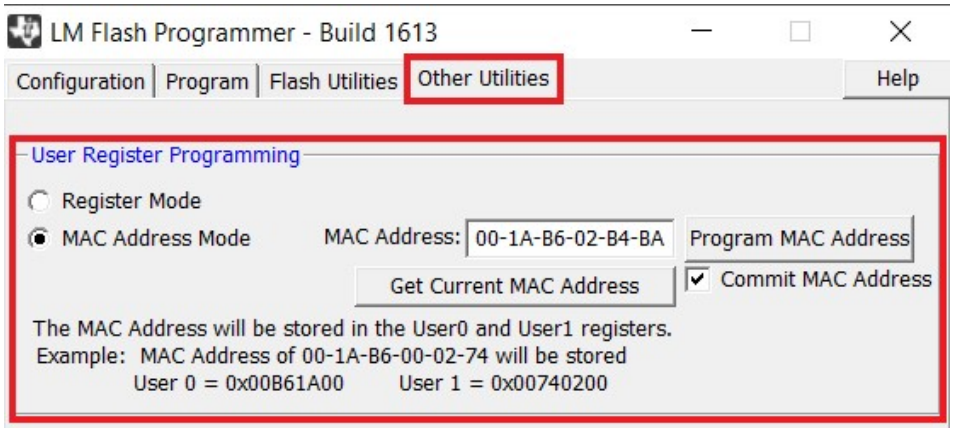


Figure 6-2. MAC Address Programming Using LM Flash Programmer

6.2.2 Using CCS

CCS also has a built-in utility to program the MAC address. CCS is most suitable if CCS is your choice of IDE for software development.

To read and program the MAC address, first go to “Tools” -> “On-Chip Flash”. The steps to read and program the MAC address will be the same as described in Section 6.2.1.

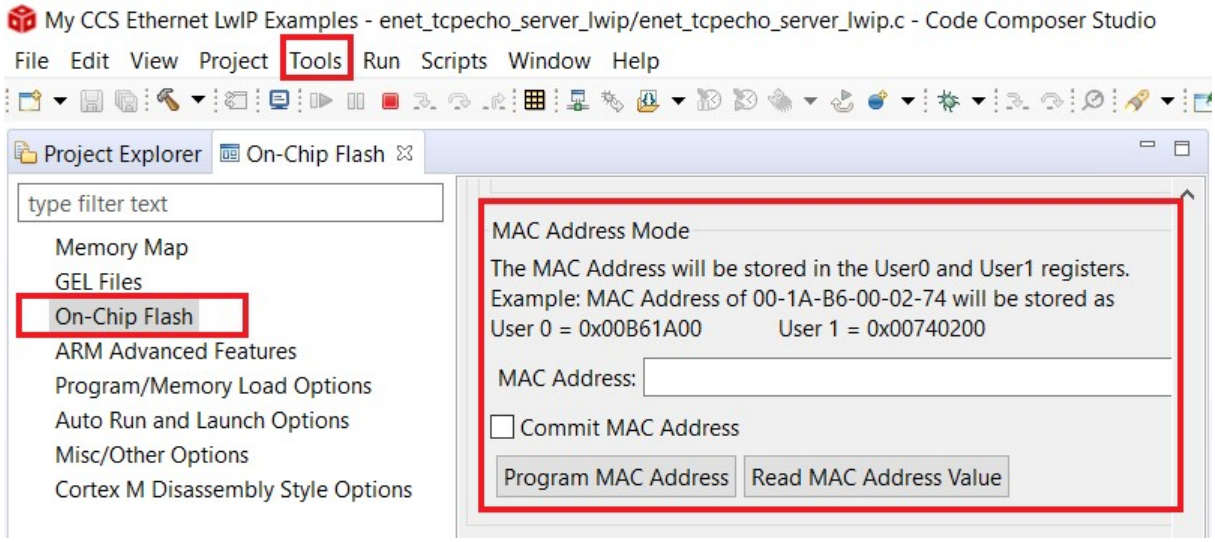


Figure 6-3. MAC Address Programming Using CCS

6.2.3 Using UniFlash

UniFlash is a TI standalone tool that supports programming of various TI devices including the MAC address for the TM4C129x MCU. UniFlash is most suitable if you are programming the MAC address on your custom board where your debug probe is not a Stellaris ICDI although the Stellaris ICDI is also supported by this tool.

To read and program the MAC address, first go to the “Settings and Utilities” tab. The steps to read and program the MAC address will be the same as described in Section 6.2.1.

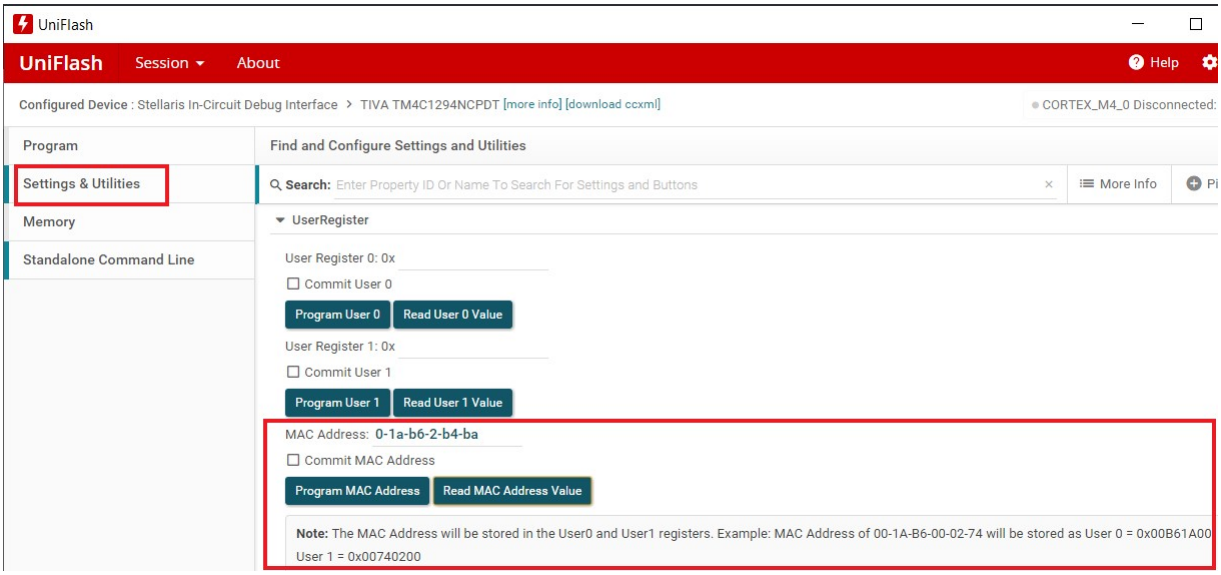


Figure 6-4. MAC Address Programming Using Uniflash

6.3 Run the enet_tcpecho_server_tirtos Example

Connect the EK-TM4C1294XL LaunchPad to either the Ethernet switch or the router with an Ethernet cable as shown in Figure 3-1. Run the example. With the CCS console window opened, you should see the IP address (pointed by arrow) displayed and the server is ready as shown in Figure 6-5. Record the IP address as you will need this information on the client side. Initially the server will be in a listening state waiting for the client to connect to it. Therefore, to continue the rest of the example, a remote client needs to be setup.

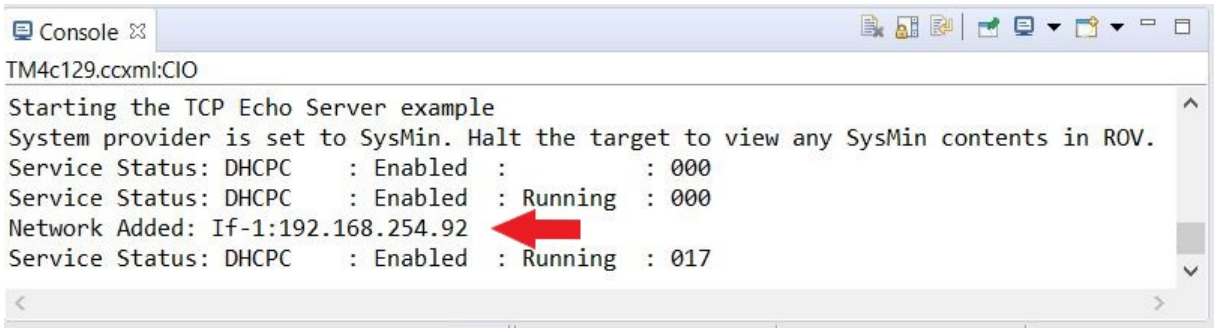


Figure 6-5. Enet_tcpecho_server_tirtos Output

Use the SocketTest tool that will act as the client running on the PC. Make sure the PC is connected to the same network as the EK-TM4C1294XL with the same subnet mask.

Follow the steps shown in Figure 6-6 to setup the client:

1. Open SocketTest and enter the server IP address as well as the port number 23. Port 23 is the default Telnet port number in TCP and UDP protocols. Finally, press the “Connect” button. In a short while the connection with the server will be established and you are ready for a conversation with the server.
2. Go to the “Message” field and type in some messages and then hit the “Send” button.
3. The message you enter will be displayed in the conversation field. When the server receives the message, it will echo back the message to the client.

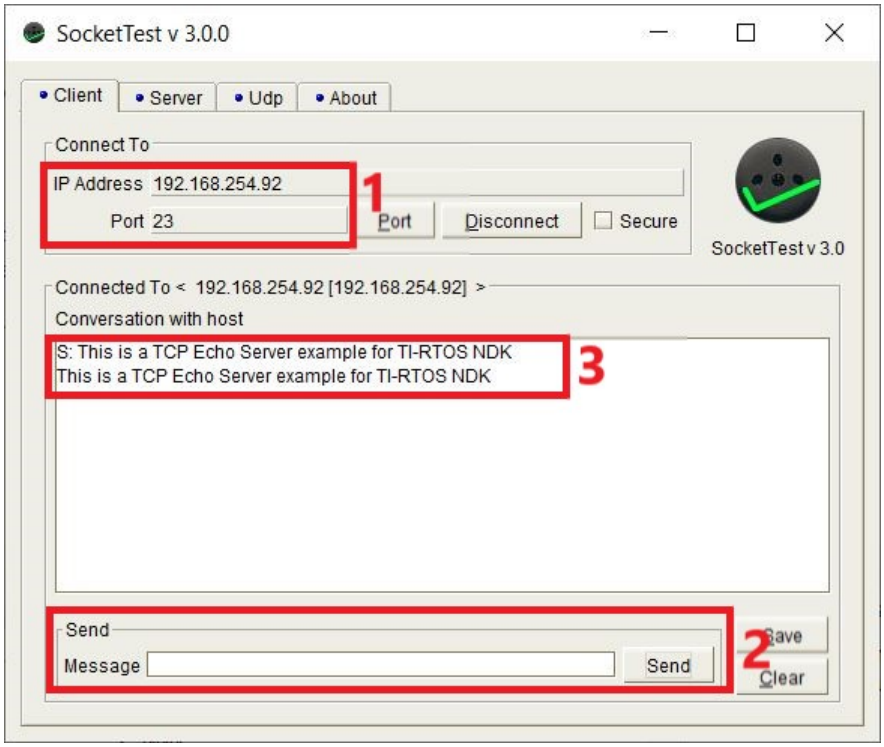


Figure 6-6. SocketTest Client Configuration for Enet_tcpecho_server_tirtos

Examine the message “This is a TCP Echo Server example for TI-RTOS NDK” that was sent by the client to the server. If you count manually, this message has a total length of 51 bytes including the two \n\r escape characters. The \n is the New Line and \r is the Carriage Return character in the ASCII table.

As can be seen in the conversation field, the server echoes back the exact message.

Also, examine the Wireshark capture in Figure 6-7 for the Ethernet traffic.

1. The client (IP Address 192.168.254.69) sends the message to the server (IP Address 192.168.254.92). For more information, see Figure 6-5 on the IP Address acquired for the server.
2. The server (IP Address 192.168.254.92) echoes back the message to the client (IP Address 192.168.254.69).
3. In Wireshark when you click on any row, it will expand with transaction details. Click the frame number 10938 to show the detail pertaining to the echoed message.
4. As you can see the length of the echoed message by the server is indeed 51 which is the same as what was sent by the client.
5. The message content echoed by the server is displayed including the two \r\n escape characters.

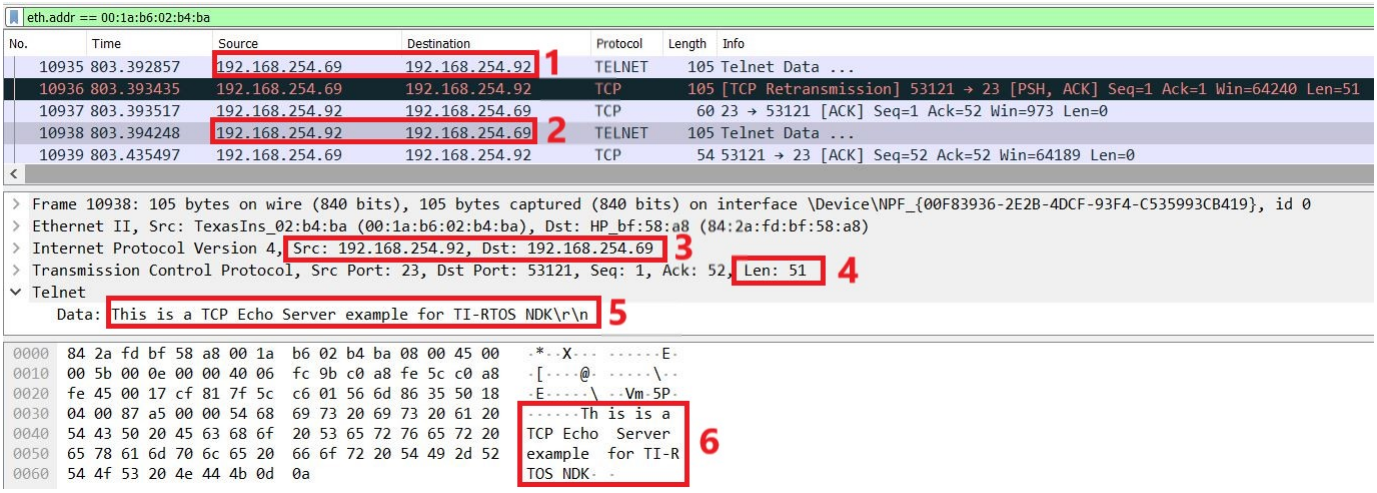


Figure 6-7. Server to Client Wireshark Capture for Enet_tcpecho_server_tirtos

7 Enet_udpecho_server_tirtos Example Overview

The User Datagram Protocol (UDP) is another well-known transport layer protocol. The enet_udpecho_server_tirtos example demonstrates an echo-server application running on the TM4C129x MCU using the UDP protocol. UDP is a connectionless protocol with no error recovery and no retransmission. The connectionless protocol is likened to mail delivery. You drop off your mail with the post office and there is no guarantee the mail will be delivered to the recipient and neither is the mail guaranteed to be delivered in its original form (damage to the mail due to rain or mishandling).

7.1 Run the enet_udpecho_server_tirtos Example

The SocketTest tool will act as the client running on the PC. Make sure the PC is connected to the same network as the EK-TM4C1294XL with the same subnet mask.

Follow the steps shown in Figure 7-1 to setup the SocketTest:

1. Go to the “Udp” tab.
2. Enter the PC’s IP address and the port number 23 and press the “Start Listening” button. The server IP address should be the address of the PC running SocketTest. To find out the PC’s IP address in your network, you can use the Windows’ ipconfig command. Bring up a Windows command window and at the prompt type “ipconfig” and you will see the IP address assigned to your PC. Note in SocketTest, the Server address field will be the address of the PC regardless if the PC is the actual server or client. SocketTest just listens for any incoming data at the specified address.
3. Enter the MCU’s IP address in the “IP Address for Client”. The IP address assigned to the MCU is shown in the CCS console window.
4. Go to the “Message” field and type in some messages and then hit the “Send” button.
5. Observe the conversation field in SocketTest.

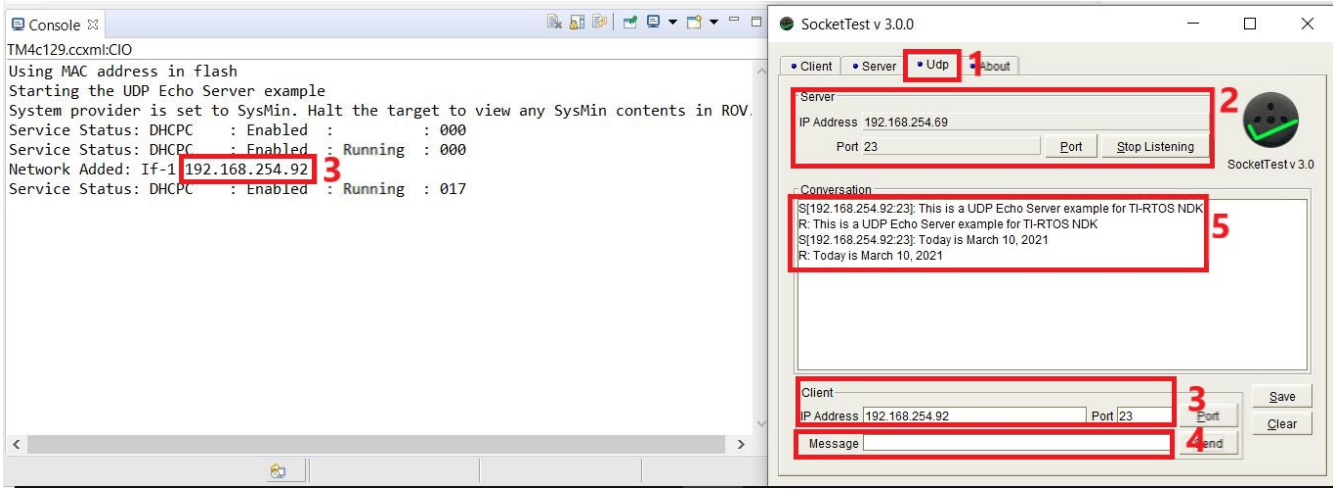


Figure 7-1. Enet_udpecho_server_tirtos Output

Examine the Wireshark capture for the UDP traffic in Figure 7-2.

1. The frame number 15351 is the first echoed message from the server (IP Address 192.168.254.92) to the client (IP Address 192.168.254.69).
2. The length is 49 bytes which is not surprising if we were to count the message “This is a UDP Echo Server example for TI-RTOS NDK\n\r” manually. Also, look at the port numbers. The port number on the PC (the client) is 60766 while the port number on the MCU (the server) is 23. The 60766 is different from what was entered in Figure 7-2. The reason is that the client never explicitly chooses a UDP port to bind to. The stack on the client side simply picks a current-available UDP port to implicitly bind the sending UDP socket to and this port may be different each time. However, the NDK stack running on the MCU is configured to bind to port 23.

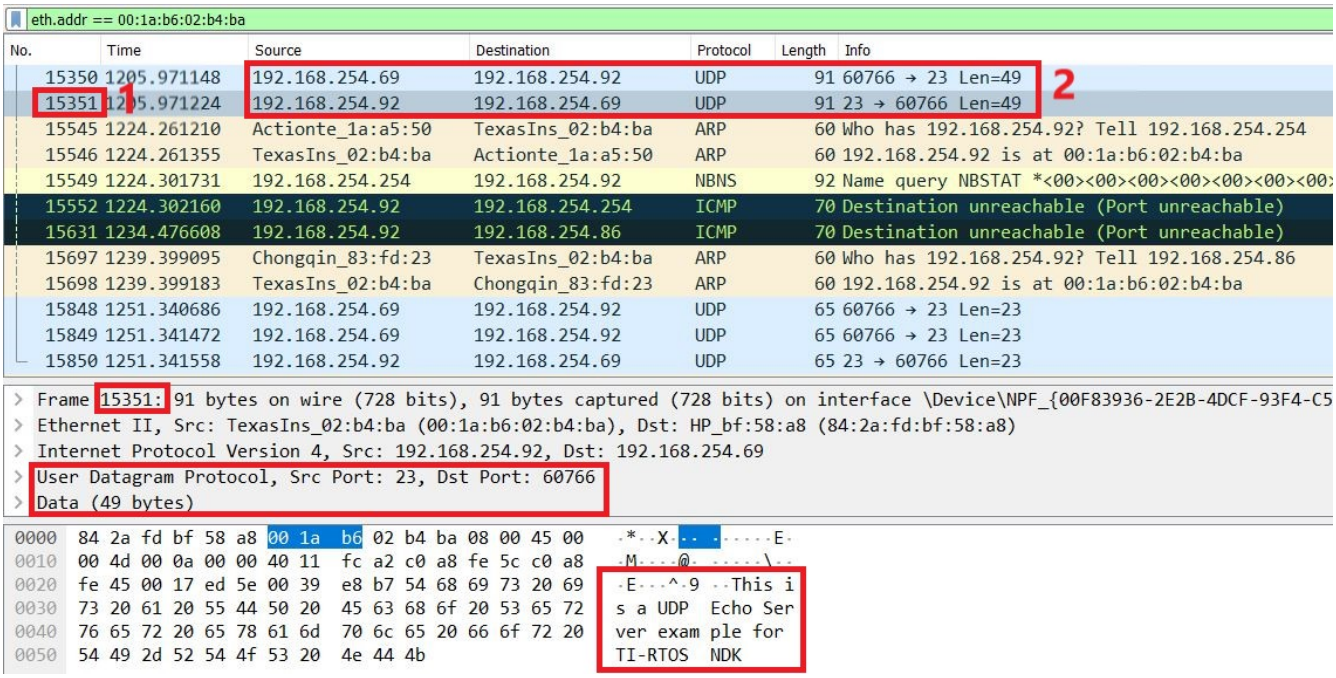


Figure 7-2. Server to Client Wireshark Capture for Enet_udpecho_server_tirtos

8 Enet_httpServer_tirtos Example Overview

This example demonstrates the EK-TM4C1294XL LaunchPad as a web server. As a web server, the EK-TM4C1294XL receives incoming network HTTP requests and sends outgoing HTTP responses along with web contents through TCP/IP connections.

The HTTP server pulls files from the embedded file system (EFS) that is included in the NDK software package's OS adaptation layer. In this example, these files are compiled into the application located on a memory-based file system.

This example also demonstrates dynamic contents generation using Common Gateway Interface (CGI). CGI is an interface specification that enables web servers to execute a program to process user requests. CGI programs execute on a web server and process input from a user. The CGI program is built from a single C callable entry-point. Each CGI function is called on its own independent task thread.

For more details, see to Web Programming with the HTTP Server.

8.1 Configure NDK for HTTP Application

For more information on how to use the XGCONF to bring up the *.cfg file for configuration, see How to Create an Ethernet Application for TI-RTOS NDK. This example already contains the httpServer.cfg that is pre-configured for an HTTP application. If you are starting with an empty *.cfg, follow the instructions below to configure NDK for an HTTP Application.

1. Add the HTTP server module to the configuration.

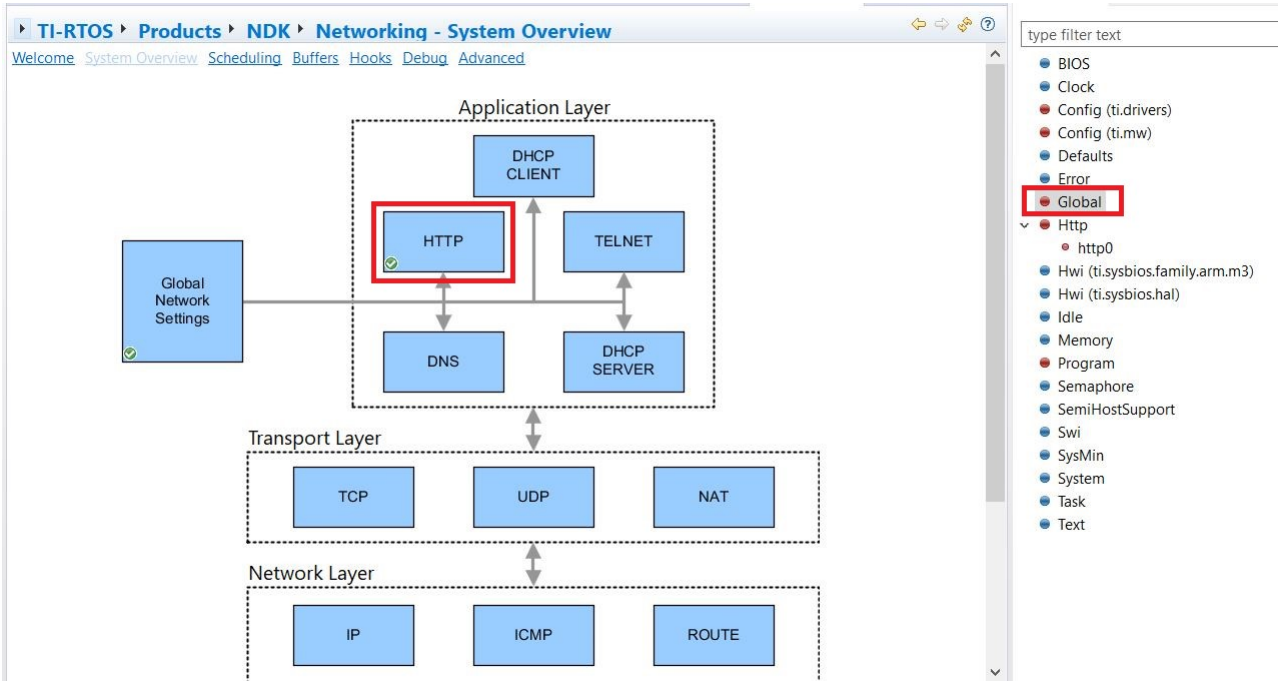


Figure 8-1. NDK Configuration for HTTP Application

2. Add HTTP instance.

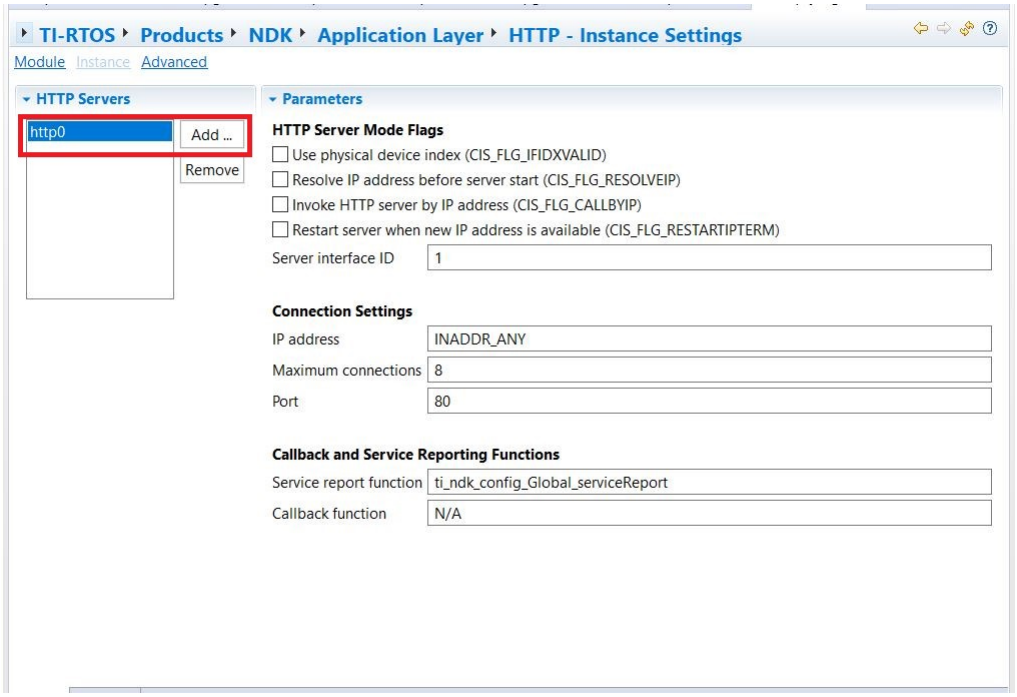


Figure 8-2. Adding a HTTP Instance

3. The default heap is too small. Select the BIOS page and navigate to the “Runtime” page to increase the heap size to 22528. The HTTP server runs in its own task. It needs a stack (2048 bytes by default). There is going to be a socket for each connected client, so that’s another 4096 bytes by default (2048 bytes each both a Rx and Tx buffer). For more details about the Networking Stack’s memory usage and ways to customize it, see https://e2e.ti.com/support/processors/f/processors-forum/947313/faq-how-is-memory-managed-in-the-ndk.

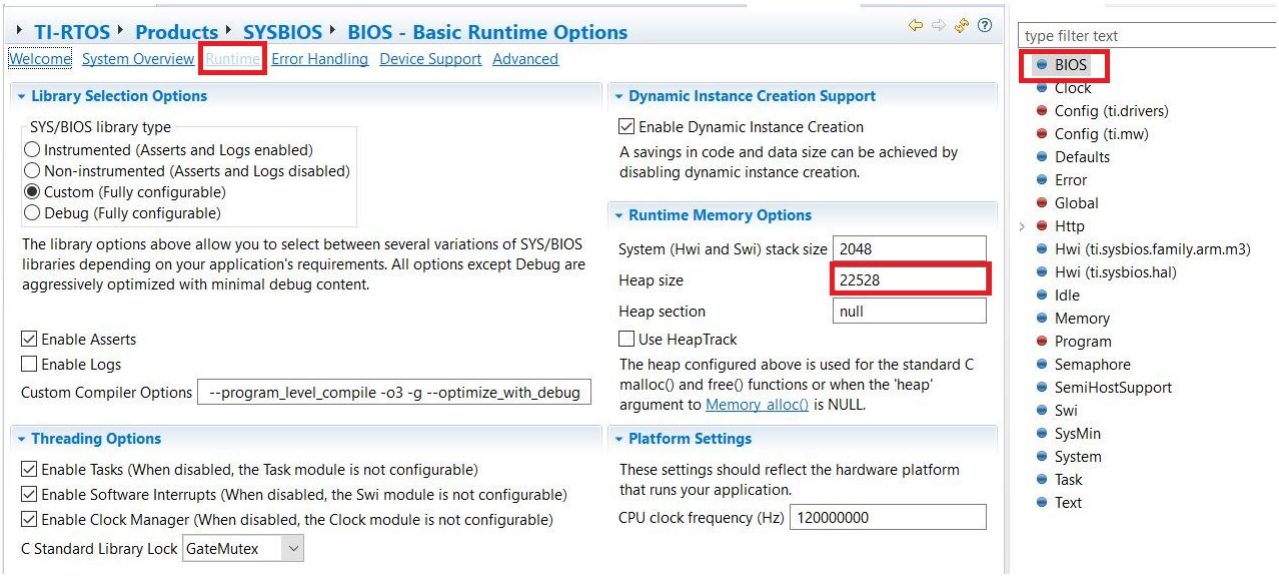


Figure 8-3. Configure the Heap Size

8.2 Embedded File System (EFS) Operation

The HTTP Server service provides a mechanism for serving HTTP content to remote HTTP client applications. It uses the Embedded File System (EFS) API contained in the OS adaptation layer. These functions in the EFS programming API include a prefix of efs_. The default installation of this API is a RAM based file system that allows the HTTP server to work on any file storage device contained in the system.

8.3 Adding HTTP Server Content

All the html pages for this example are stored in the “fs” directory of the project, see Figure 8-4. These html files form the web pages to be served by the web server.

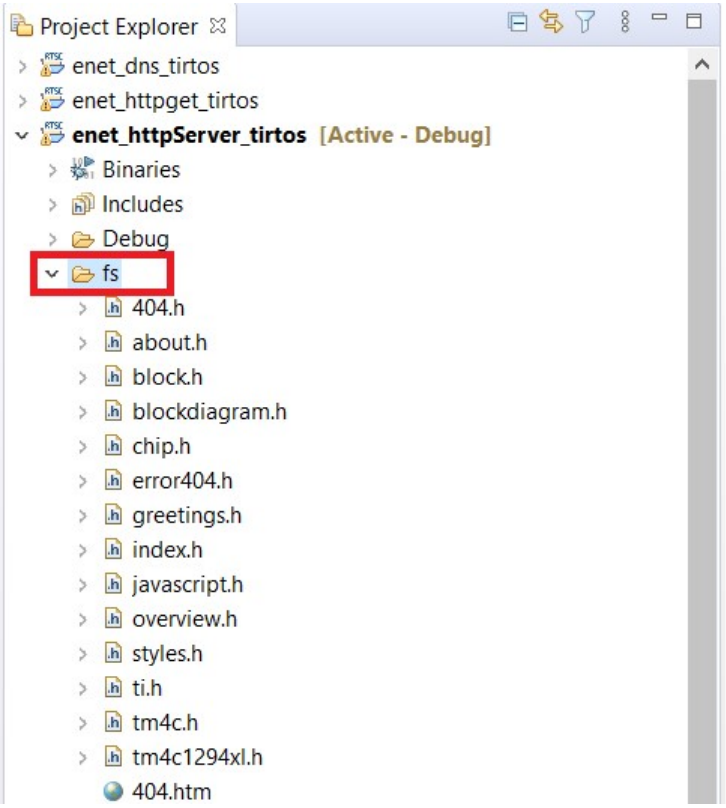


Figure 8-4. File System Directory

The html web pages must be first converted from their binary HTML files into data arrays declared in C. An MS-DOS utility “binsrc” is supplied in the NDK package to allow conversion of files to a C array. The binsrc utility can be found in <TI-RTOS Installation>\products\ndk_2_25_00_09\packages\ti\ndk\tools\binsrc\binsrc.exe.

The calling format for binsrc is:

```
>binsrc <input file name> <output file name> <identifier>
```

Parameters:

- Input file name: File to be converted
- Output file name: Name for file containing C data representation of the input file name
- Identifier: C name for data

For example, to convert an HTML file index.html for use by EFS, the following command could be executed from the Windows command window:

```
>binsrc index.html index.h INDEX
```

The index.html after conversion to index.h would contain the following as shown in Figure 8-5.

1. The index.h file defines the size of the file to be 1932 bytes. Later we will look at Wireshark to confirm that the web server indeed transfers the same number of bytes in its HTTP response to the client.
2. The INDEX character array begins with <!DOCTYPE HTML>. Open a text editor to read the first line of index.html and you can confirm that it is the same as what was stored in the INDEX array.
3. Look at the content of the next few bytes in the INDEX array and it encodes <!—Copyright © 2013-2021 Texas Instruments Incorporated. All rights…..>. Again, compare it with the index.html and confirm that index.h is a binary representation of the index.html.

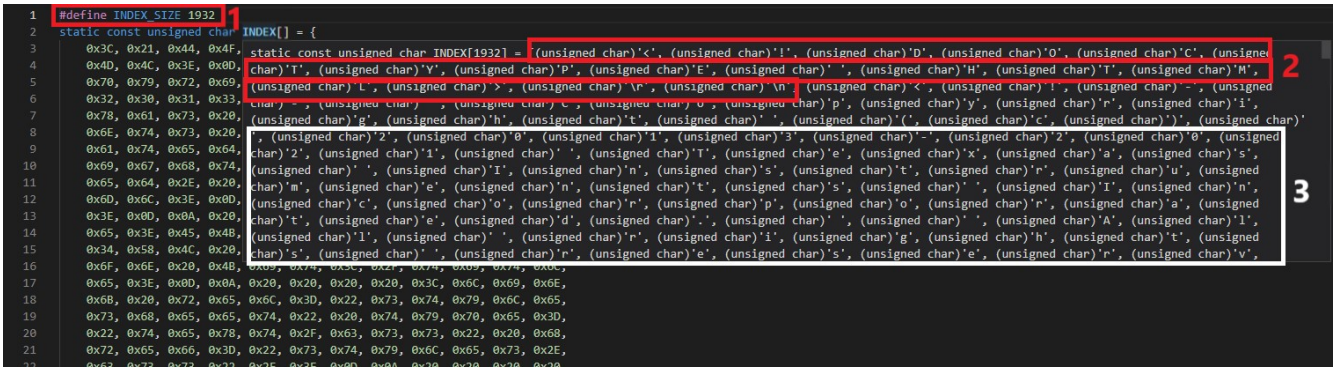


Figure 8-5. Index.html Binary File Content

8.4 Declaring HTML Files to EFS

Once the HTML file is converted to a memory image, the file is declared to the EFS file system by calling the function *efs_createfile()*. All the HTML files are typically created at the same time, during initialization, before the HTTP server is actually invoked. In the enet_httpServer_tirtos example code, there are two functions used, *AddWebFiles()* and *RemoveWebFiles()*. These functions include all the code necessary to initialize and clean up the EFS file environment.

Below is the snippet of example code to declare HTML files to EFS.

```
/* file system header file */
#include "fs/index.h"
#include "fs/about.h"
#include "fs/overview.h"
Snip…

Int getTime(SOCKET s, int length)
{
    Char buf[200];
    static UInt scalar = 0;
Snip…

Void AddWebFiles(Void)
{
    efs_createfile("index.html", INDEX_SIZE, (UINT8 *)INDEX);
    efs_createfile("overview.htm", OVERVIEW_SIZE, (UINT8 *)OVERVIEW);
    efs_createfile("about.htm", ABOUT_SIZE, (UINT8 *)ABOUT);
Snip…
    efs_createfile("getTime.cgi", 0, (UINT8 *)&getTime);
Snip…


Void RemoveWebFiles(Void)
{
    efs_destroyfile("index.html");
    efs_destroyfile("overview.htm");
    efs_destroyfile("about.htm");
Snip…
    efs_destroyfile("getTime.cgi");
Snip…
```

Figure 8-6 shows where the two hook functions *AddWebFiles()* and *RemoveWebFiles()* are declared in the NDK configuration.

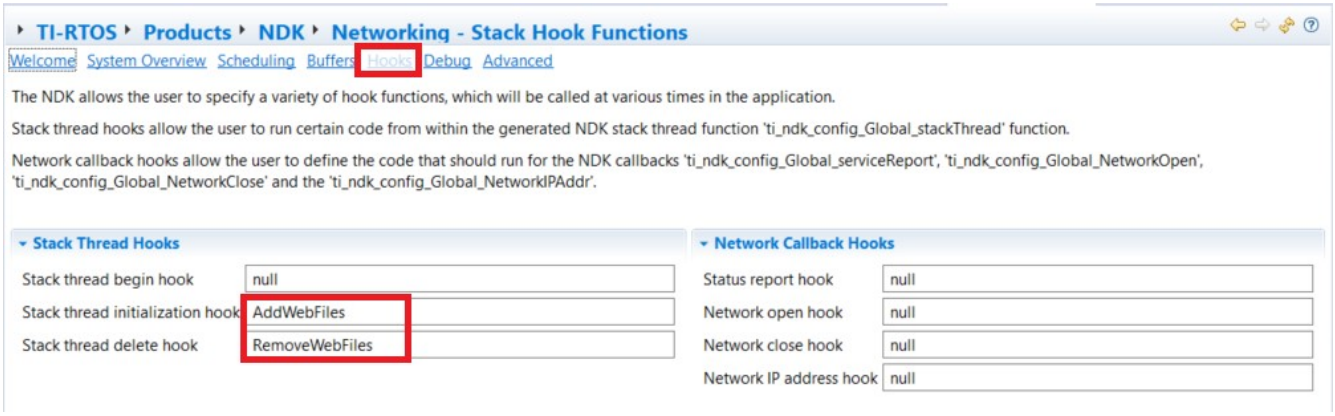


Figure 8-6. HTTP Hooks Declaration

Once the above code is run, the EFS system is ready for the HTTP server to serve up the content.

8.5 Writing CGI Functions

CGI programs must be in the EFS for the HTTP server to see them. See the above example code where an entry for the file getTime.cgi that translated into the C function *getTime()*. Whenever a HTTP POST is made to the file getTime.cgi, the *getTime()* function is called.

For details on writing CFG functions, see Writing CGI Functions.

8.6 Run the enet_httpServer_tirtos Example

Rebuild the example if you create your own web contents or load and run the existing example. In the CCS console window, you should see the web server IP address displayed. For more information, see Figure 6-5.

Once the web server is running, enter the IP address on your browser's URL field as shown in Figure 8-7 and the example web page is rendered on the browser. The index.html normally is the default page shown on a web page if no other page is specified. In other words, index.html is the name used for the homepage of the website.

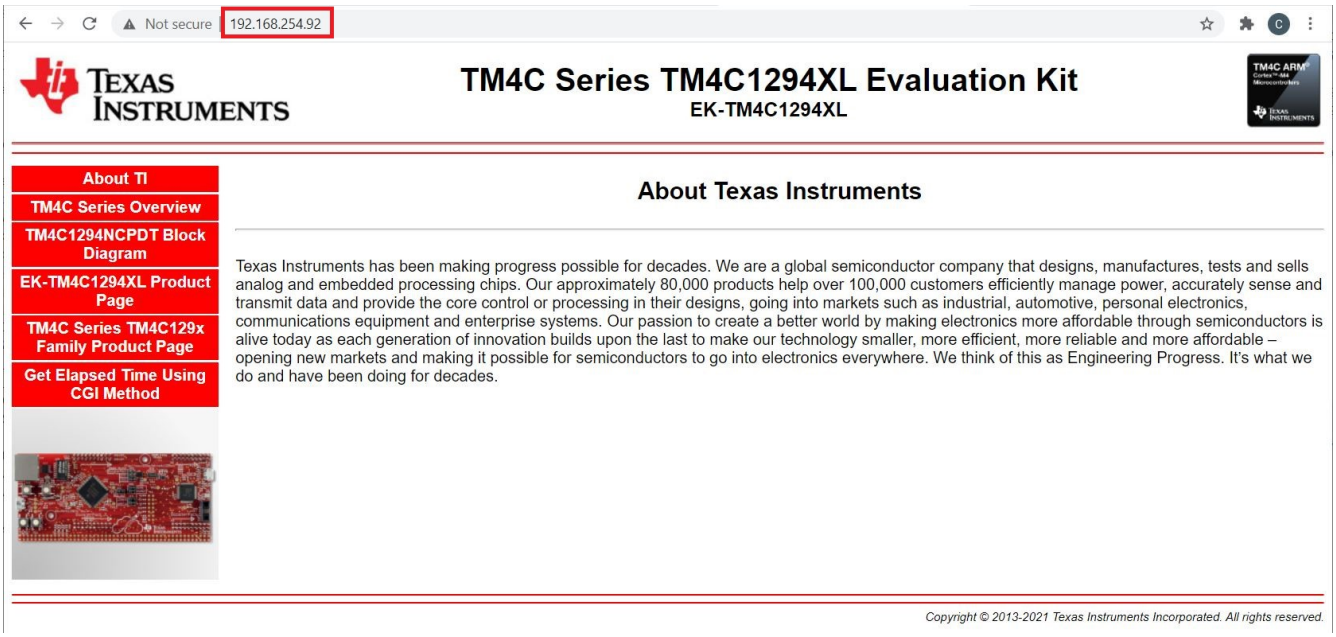


Figure 8-7. Enet_httpServer_tirtos Web Server Home Page

Several hyperlinks and images are displayed on the home page. Click any one of them to get a feel of how they respond. For example, click the “TM4C1294NCPDT Block Diagram” and you will see the web page displayed as shown in Figure 8-8. Note that you could also reach the same web page by directly typing “192.168.254.92/block.htm” on the URL.

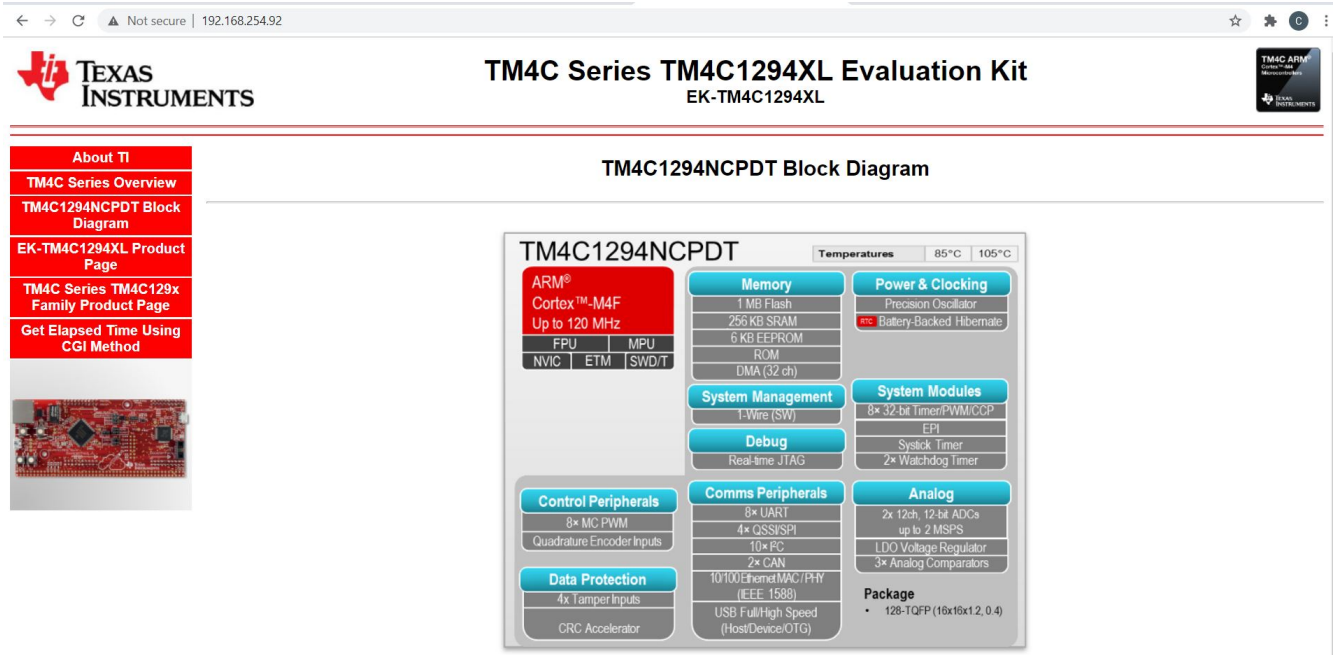


Figure 8-8. Block Diagram Page

Examine the Wireshark capture in Figure 8-9.

1. The first HTTP GET request from the client happens on frame 4460 and the corresponding server response happens on frame 4468. The server responds with an OK response along with the contents of the web page which is the default index.html.
2. The file size is 1932 bytes. This is exactly the same as the INDEX array specified in the index.h file, shown in Figure 8-5.
3. There are a total of 54 lines of html data with the first line as <!DOCTYPE HTML>\r\n. This is matching Figure 8-5.
4. Again, the second line of index.html is <!—Copyright (c) ……> which is the same as what was encoded in the INDEX character array.

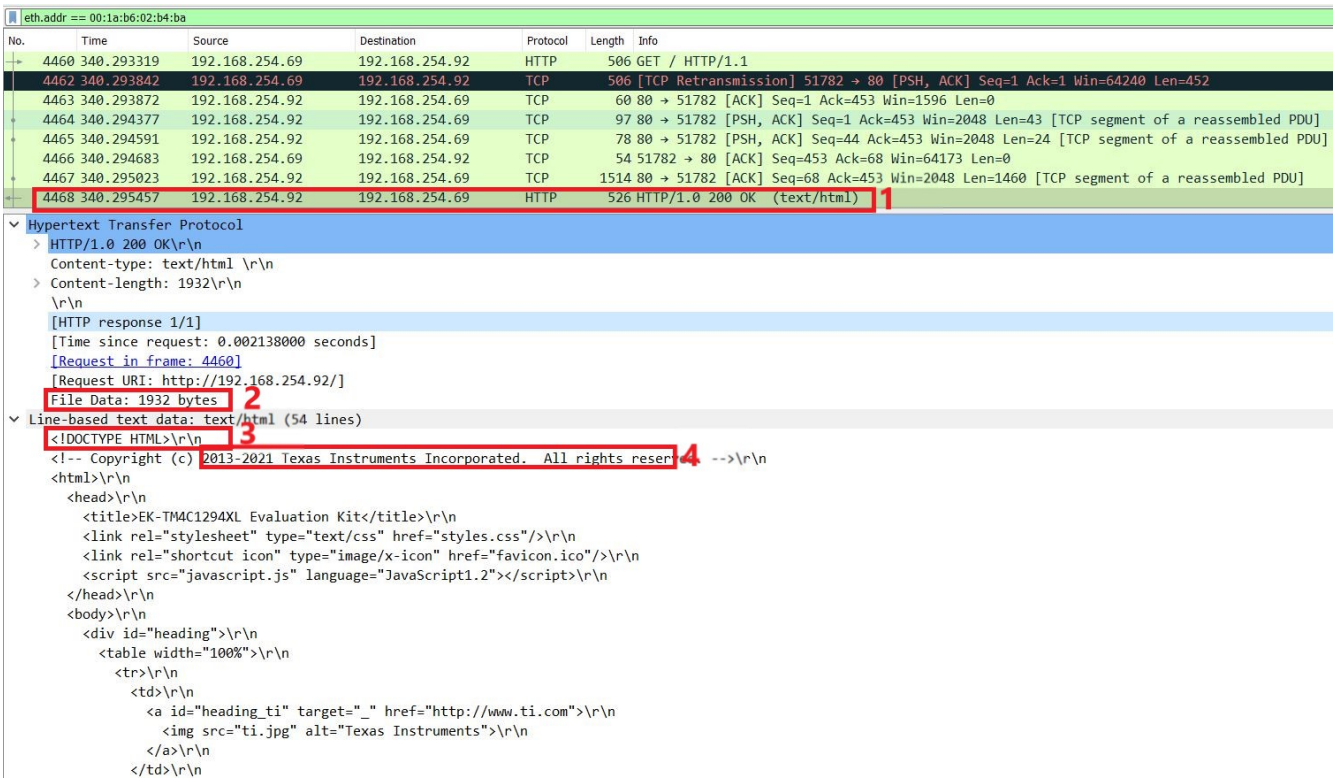


Figure 8-9. Wireshark Capture for enet_httpServer_tirtos Web Server

Finally, from the home page, click the “Get Elapsed Time Using CGI Method”. This action triggers the *getTime()* function to execute and print out the elapsed time the web server has been active, see Figure 8-10. Press the “Refresh” button on your browser to refresh the screen to see an updated elapsed time. Note that you can also trigger the CGI function by directly specifying it in the URL as in “192.168.254.92/getTime.cgi”.

Figure 8-10. Web Server Elapsed Time

9 Enet_dns_tirtos Example Overview

Every device connected to a network will have a unique IP address which relies on the IP (Internet Protocol) for communications. However, the IP address is either a 32-bit address as in IPv4 or 128-bit as in IPv6 which is hard to remember by humans. The DNS (Domain Name System) is an application layer service that translates the “human-friendly” domains names to IP addresses. It is easy for humans to remember www.ti.com as opposed to its numerical IP address. Note the DNS is rather an application service that relies on the UDP protocol rather than a protocol per se.

This simple example demonstrates using the DNS feature to retrieve the IP addresses of four different websites.

9.1 How to Configure NDK for DNS

To use DNS, the DNS module in NDK must be enabled. Since DNS relies on UDP as the transport protocol, the UDP module needs to be enabled as well.

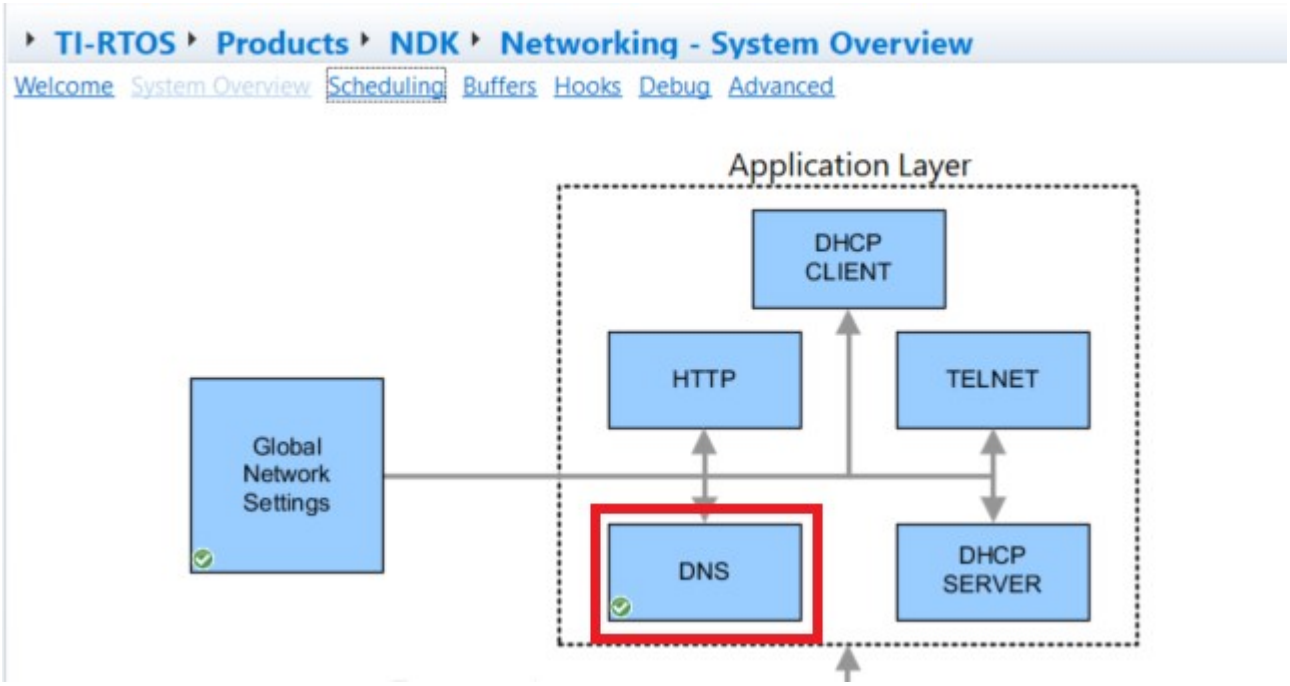


Figure 9-1. Configure NDK for DNS

9.2 How to View the DNS Traffic on Wireshark

If you hook up the MCU device and the PC to the Ethernet switch, you will not be able to view the DNS traffic on Wireshark. The reason is that the DNS traffic is between the DNS server and the device. The switch will not route the traffic to the PC running Wireshark. In order to view the DNS traffic, you either need to have an Ethernet hub which broadcasts all traffic on the network to all ports connected to it or you need to configure the Ethernet switch for “Port Mirroring”. An Ethernet hub is hard to find these days. It is easy to find a Smart Switch that you can configure for Port Mirroring. Figure 9-2 shows where the EK-TM4C1294XL connecting to the switch Port 2 is mirrored onto Port 1 that is connected to the PC. With port mirroring, all traffic to/from Port 2 will be snooped by the Wireshark on port 1. Consult your switch or router data sheet on how to configure port mirroring.

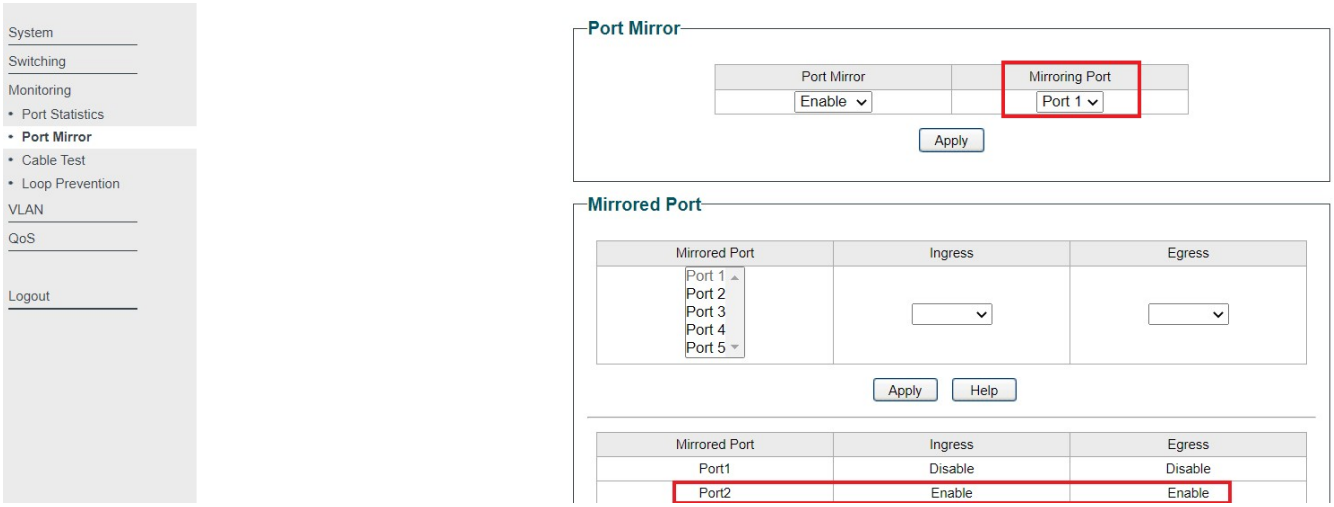


Figure 9-2. Port Mirroring

9.3 Run the enet_dns_tirtos Example

As expected, running this example prints the four IP addresses of the corresponding websites on the CCS console window, see Figure 9-3. For more details, look at the Wireshark capture particularly for www.google.com.

- In the highlighted box 1 and box 2 of the Wireshark capture in Figure 9-4, four DNS packets are observed corresponding to the four websites.
- The source address of the transaction is from the IP address 192.168.254.254. This happens to be the IP address of the router to which this application is connected. A smart router normally will act as the DNS server and store the past visited domain names in its cache for fast retrieval.
- In box 3, the returned IP address for www.google.com is 172.217.1.132. You can confirm by typing this address on your browser’s URL field and it should lead you to the Google website.
- In box 4, the IP address is expressed as a 32-bit binary value equal to 0xACD90684. The 0xAC is equal to decimal 172, the 0xD9 is equal to decimal 217 and likewise for the rest.

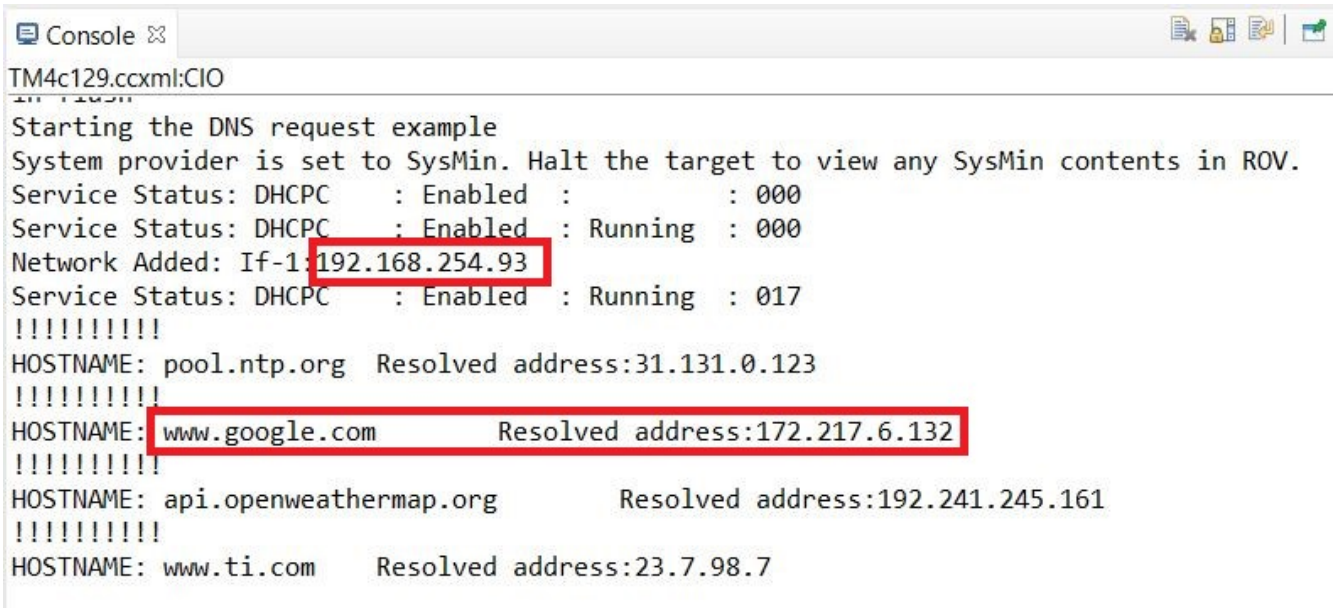


Figure 9-3. Enet_dns_tirtos Output

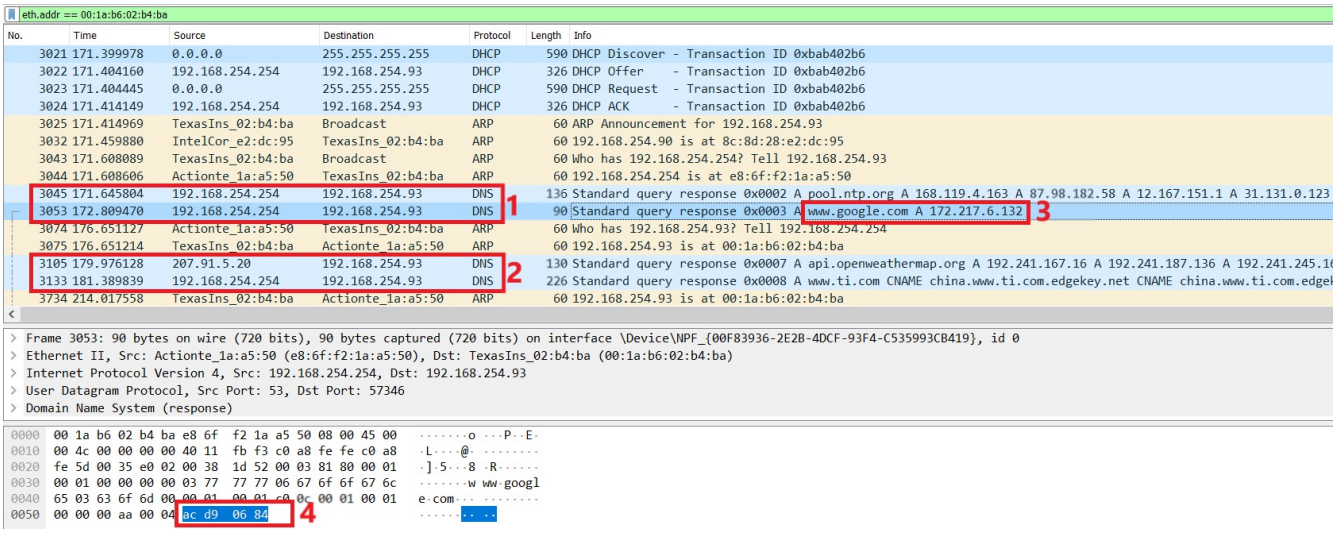


Figure 9-4. Wireshark Capture for Enet_dns_tirtos

10 Enet_sntp_tirtos Example Overview

Simple Network Time Protocol (SNTP) is a simplified version of NTP. SNTP typically provides time within 100 mS of the accurate time without the complex filtering mechanisms of NTP. SNTP is an application layer protocol that is based on UDP as the transport layer.

This example acquires the network time using the SNTP protocol and displays the current time adjusted for North American Central Time (CT) Zone on the CCS console window.

10.1 Run the enet_dns_tirtos Example

The client sends the request to the pool.ntp.org website from which a randomly selected public time server is chosen to provide the network time as shown in the Wireshark capture in Figure 10-2. The acquired network time is adjusted for Central Time Zone before displaying on the CCS console window shown in Figure 10-1.

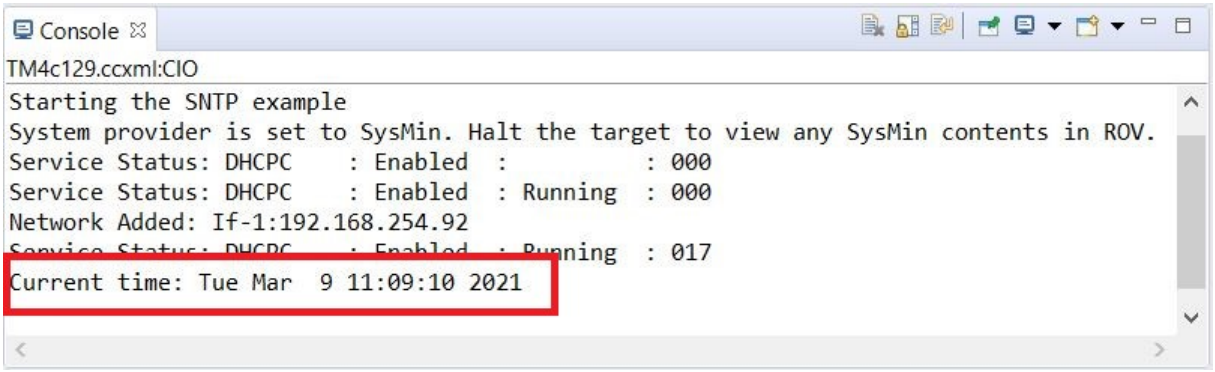


Figure 10-1. Enet_sntp_tirtos Output

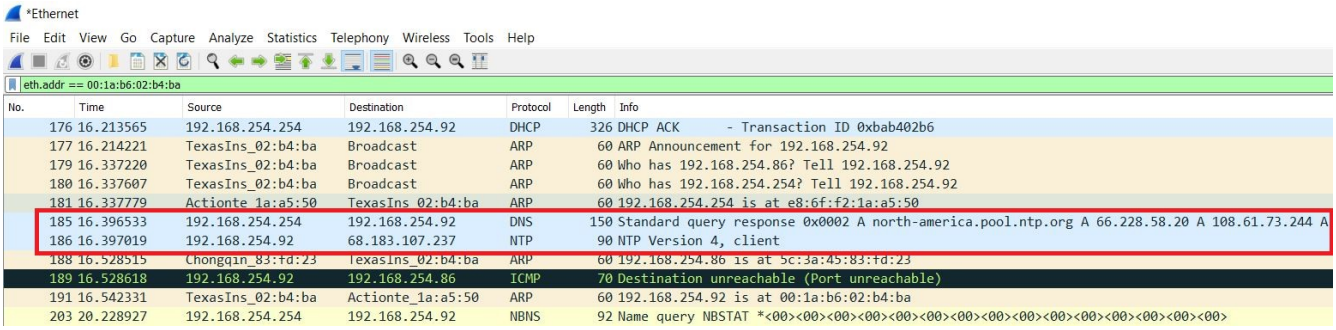


Figure 10-2. Wireshark Capture for Enet_sntp_tirtos

11 Enet_tcpecho_client_tirtos Example Overview

The enet_tcpecho_client_tirtos example demonstrates a client application that first connects to the server with a greetings message “Hello from TM4C1294XL Connected LaunchPad\n” and then echoes back whatever it receives from the server.

As illustrated in the BSD socket flowchart for TCP in Figure 1-3, the client will use connect() to connect to the specified server address and port. Once the connection is established, the client will use recv() for receiving data from the server and then echo the data back.

Another difference between the client and the server in the flowchart is that the client does not need to call bind(). Binding is normally not needed on the client side for TCP. There may be circumstances where binding the client is needed in which you will use bind() to bind the client. An example would be that a firewall on the client that only allows outgoing connections on a certain port.

11.1 Configure the Server IP Address

To run this example, the Server IP address and port number must be known during compile time so the client will make the connection with the server.

- Define the SERVER_IPADDR and SERVER_PORT in the tcpEchoClient.c file. Note that the below address is only an example. You must change the server IP address to that you are connecting to in your network, otherwise the example will not work.

```
#define SERVER_IPADDR "192.168.254.69"
#define SERVER_PORT 23
```

Recompile the project and you are ready to run the example.

11.2 Configure the SocketTest Server

Since this is a client application, the server must be setup first and be in a listening state before the client can connect to it.

Follow the steps shown in Figure 11-1 to setup the SocketTest server:

1. Open the Server tab in SocketTest.
2. Enter the server IP address as well as the port number. The IP address is the one of the PC which SocketTest is running on. The IP and port number need to match the settings defined in tcpEchoClient.c as shown in Section 11.1. Finally, press the “Start Listening” button. Wait for the greeting message to appear on the conversation window once the client connects.

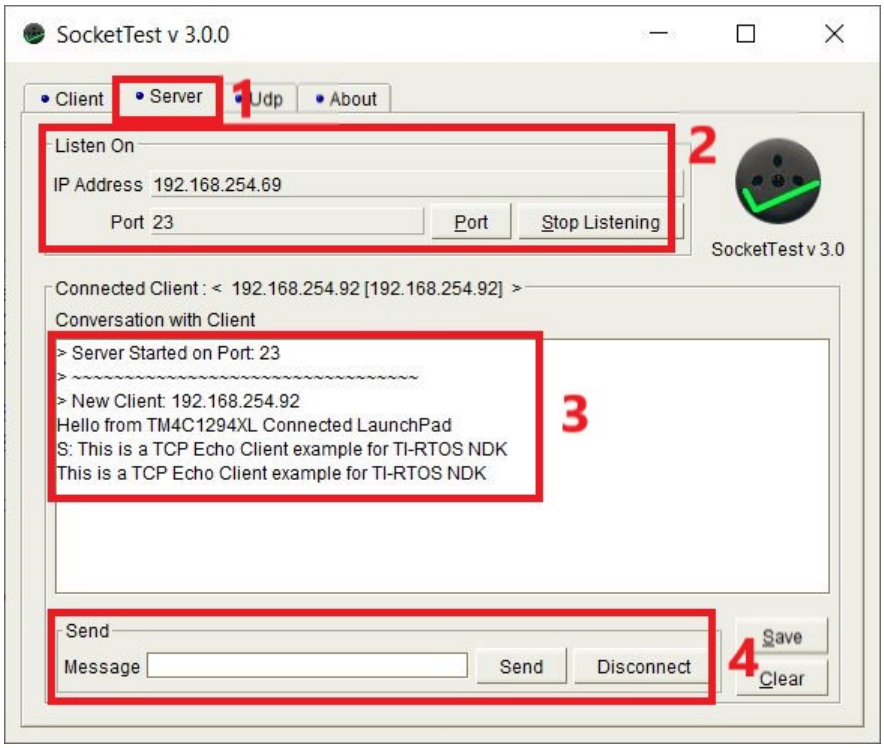


Figure 11-1. SocketTest Server Configuration for Enet_tcpecho_client_tirtos

11.3 Run the enet_tcpecho_client_tirtos Example

After the example has run, it will display the client IP address acquired from the DHCP server on the CCS console window. Once the client connects to the SocketTest server, it will send a “Hello from TM4C1294XL Connected LaunchPad\n” greeting message to the server.

Look at the conversation field in box 3 of Figure 11-1 where it displays the “Hello from TM4C1294XL Connected LaunchPad\n” message from the client IP address 192.168.254.92 as soon as the server accepts the connection. Go to the Message field in the SocketTest shown in box 4 and type some message. Whatever message is entered is echoed by the client. In this example, the message entered is “This is a TCP Echo Client example for TI-RTOS NDK\n\r” with a total length of 53 characters.

Examine the Wireshark capture in Figure 11-2.

1. As discussed previously, TCP is a connection-based protocol. Here you see the SYN segment sent by the client 192.168.254.92 to the server 192.168.254.69 to establish a connection and the ACK segment from the server to accept the connection.
2. After the connection is accepted, the client sends the “Hello from TM4C1294XL Connected LaunchPad\n” message. This is shown as Telnet Data in frame 129.
3. Notice the length of the message is 42 bytes which matches the total number of characters in the greetings message.
4. The content of the message captured is matching the greeting that was sent by the client.

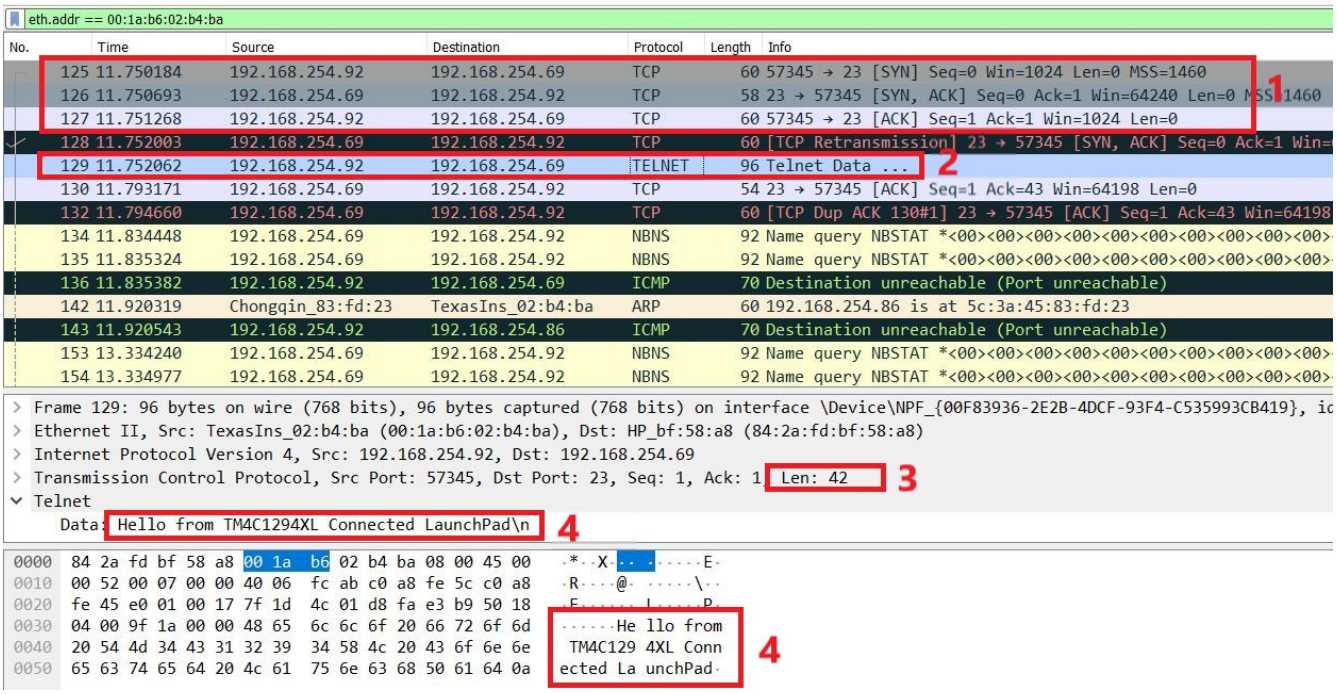


Figure 11-2. Client Server Wireshark Capture for Enet_tcpecho_client_tirtos

12 Enet_udpecho_client_tirtos Example Overview

The enet_udpecho_client_tirtos example is a client application using UDP protocol. It is similar to the server in terms of APIs as depicted in the UDP flowchart. This example will use sendto() to send a greeting message "Hello from TM4C1294XL Connected LaunchPad\n\r" to the remote server. It will then echo anything that it receives from the server.

12.1 Run the enet_udpecho_client_tirtos Example

For information on how to setup SocketTest for UDP testing, see Section 7.1. SocketTest in Figure 12-1 first shows the greeting message “Hello from TM4C1294XL Connected LaunchPad\n\r” sent by the client. The server then sends a message “This is a UDP Echo Client example for TI-RTOS NDK\n\r”. The message is received by the client and then echoed back to the server. The messages captured by the Wireshark in Figure 12-2 can be cross-referenced.

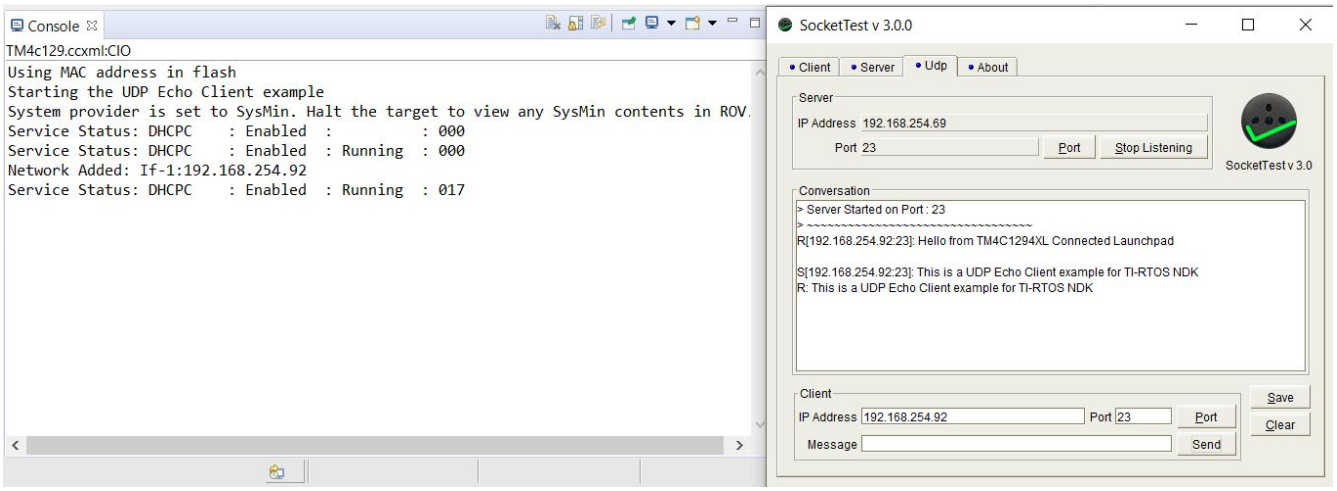


Figure 12-1. Enet_udpecho_client_tirtos Output

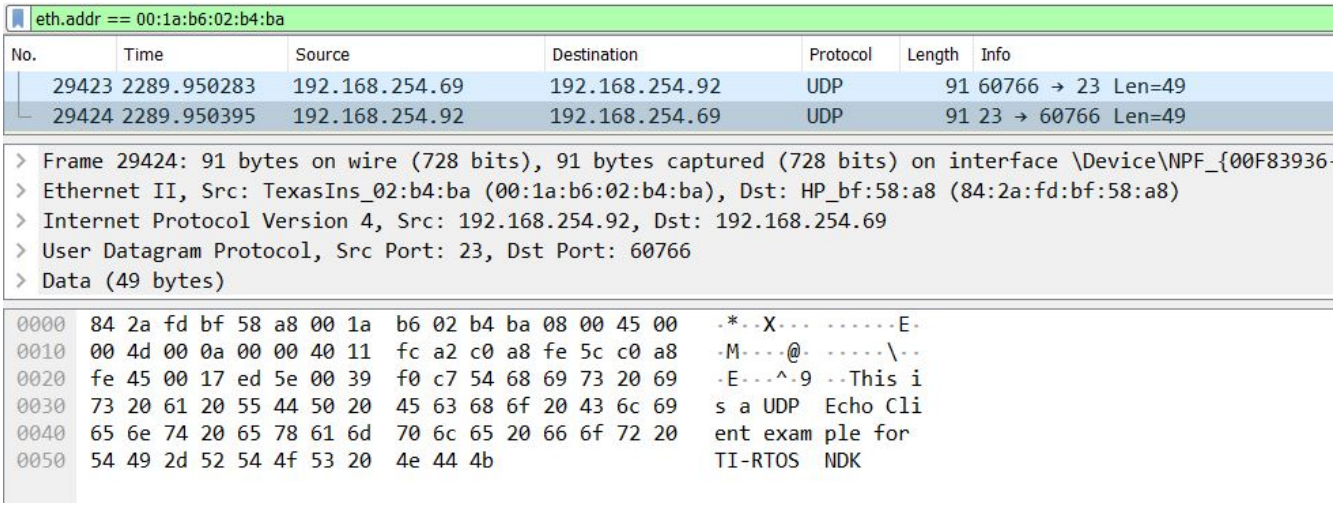


Figure 12-2. Client Server Wireshark Capture for Enet_udpecho_client_tirtos

13 Enet_httpget_tirtos Example Overview

The enet_httpget_tirtos example makes an HTTP GET call to www.example.com using the request URI. The response status code, header fields and body from the HTTP server are processed to get response status and data. The HTTP response status and the number of bytes of data received are printed on the CCS console window.

13.1 How to Configure NDK for HTTP GET Example

The HTTP application relies on TCP as the underlying transport protocol. The only NDK module that is needed for this example is the TCP module. To enable the TCP module if you are starting from an empty .cfg project.

It is also worth noting that a client must first obtain its IP address before making a connection and an HTTP request to the server. If the client attempts to connect to the server without an IP address, then the Address Resolution Protocol (ARP) will fail. ARP is a communication protocol used for discovering the link layer address, such as a MAC address, associated with a given IP address. The NDK configuration in this example ensures that the callback hook function is only called after the IP address is obtained. *netIPAddrHook()* is associated as the user defined hook function for the NDK Network IP address callback function *NetworkIPAddr()* that is called when an IP address is added from the system, see Figure 13-1. When the *netIPAddrHook()* function is called, it starts the HTTP request task.

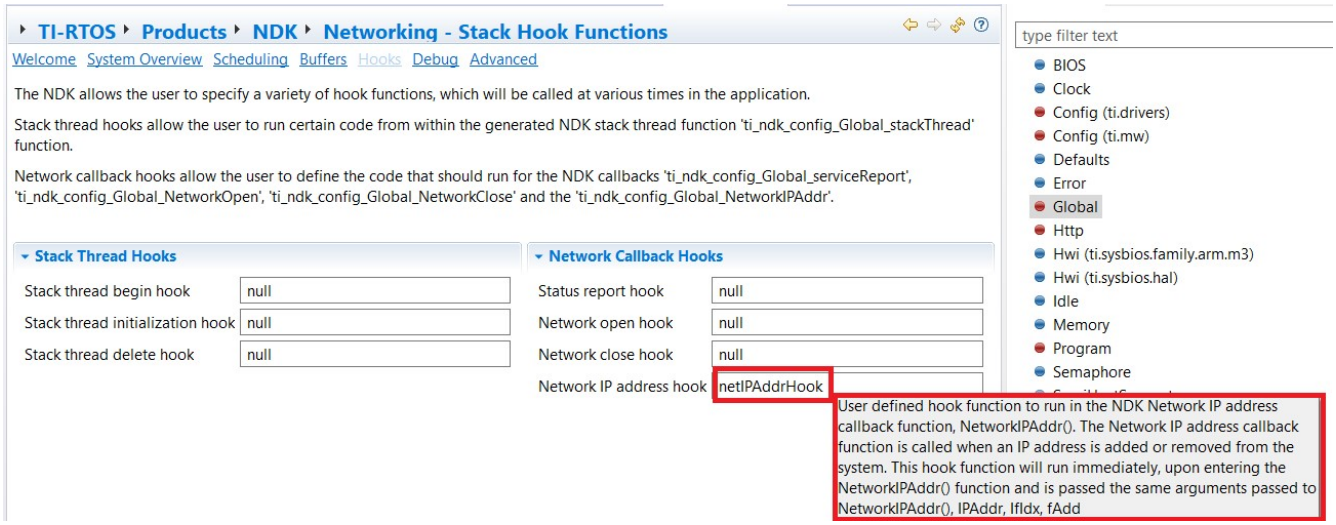
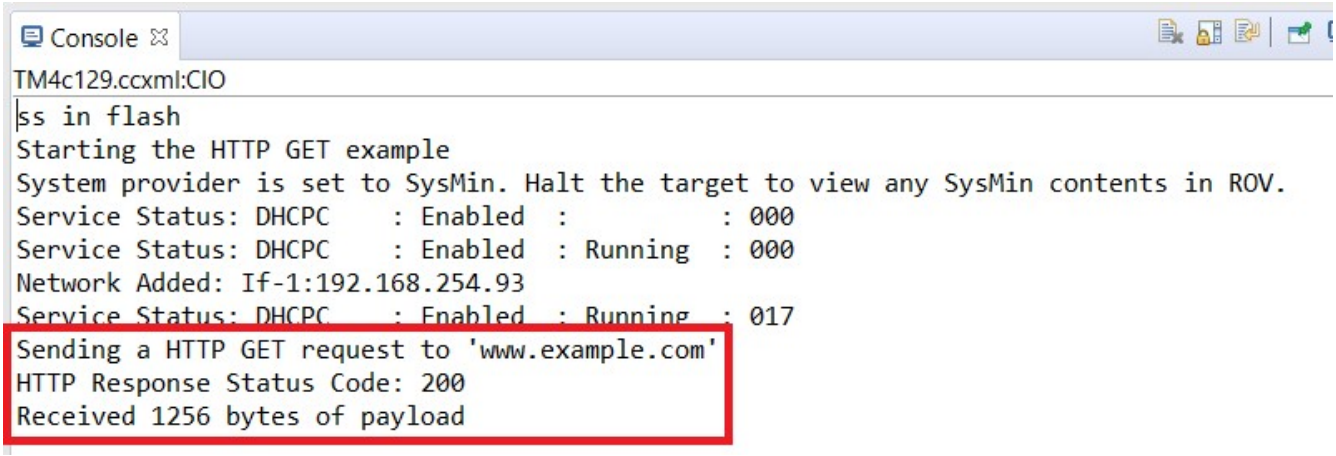


Figure 13-1. Network IP Address Hook Function

13.2 Run the enet_httpget_tirtos Example

Figure 13-2. Enet_httpget_tirtos Output



14 References

- Texas Instruments: *TI Network Developer's Kit (NDK) User's Guide*
- Texas Instruments: *TI Network Developer's Kit (NDK) API Reference*
- TI-RTOS-MCU Overview
- Texas Instruments: *TI-RTOS for TivaC Getting Started Guide*
- Texas Instruments: *TI-RTOS User's Guide*
- Texas Instruments: *SYS/BIOS (TI-RTOS Kernel) User's Guide*

IMPORTANT NOTICE AND DISCLAIMER

TI PROVIDES TECHNICAL AND RELIABILITY DATA (INCLUDING DATASHEETS), DESIGN RESOURCES (INCLUDING REFERENCE DESIGNS), APPLICATION OR OTHER DESIGN ADVICE, WEB TOOLS, SAFETY INFORMATION, AND OTHER RESOURCES “AS IS” AND WITH ALL FAULTS, AND DISCLAIMS ALL WARRANTIES, EXPRESS AND IMPLIED, INCLUDING WITHOUT LIMITATION ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT OF THIRD PARTY INTELLECTUAL PROPERTY RIGHTS.

These resources are intended for skilled developers designing with TI products. You are solely responsible for (1) selecting the appropriate TI products for your application, (2) designing, validating and testing your application, and (3) ensuring your application meets applicable standards, and any other safety, security, or other requirements. These resources are subject to change without notice. TI grants you permission to use these resources only for development of an application that uses the TI products described in the resource. Other reproduction and display of these resources is prohibited. No license is granted to any other TI intellectual property right or to any third party intellectual property right. TI disclaims responsibility for, and you will fully indemnify TI and its representatives against, any claims, damages, costs, losses, and liabilities arising out of your use of these resources.

TI’s products are provided subject to TI’s Terms of Sale (https:www.ti.com/legal/termsofsale.html) or other applicable terms available either on ti.com or provided in conjunction with such TI products. TI’s provision of these resources does not expand or otherwise alter TI’s applicable warranties or warranty disclaimers for TI products.

Mailing Address: Texas Instruments, Post Office Box 655303, Dallas, Texas 75265