Freescale Semiconductor
Application Note

AN2836
Rev. 0, 9/2004

Web Server Development with MC9S12NE64 and OpenTCP

By: Steven Torres
8/16 Bit System Engineering
Austin, Texas

Introduction

Ethernet connectivity of embedded devices is a growing trend in industrial and consumer applications. Ethernet is a medium of choice because of its competitive performance, relatively low price of implementation, established infrastructure, and inter-operability. Ethernet is also easy to use, widely available, and scalable. Ethernet is described by IEEE Standard 802.3™.

With Ethernet and TCP/IP data transmission, embedded devices can be connected to the Internet, which allows access to the embedded device from across the world. Figure 1 shows a simplified illustration of an embedded device that is connected to a remote host via the Internet. Figure 1 shows that the embedded device and remote host can operate on different networks, but the connection between the devices is transparent.

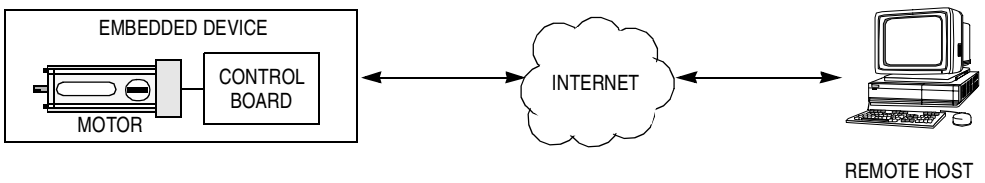

Figure 1. Embedded Device on Internet

This product incorporates SuperFlash® technology licensed from SST.

Acronyms and Terms

Table 1. Acronyms and Terms

Acronym/ Term	Description	Definition
ARP	Address resolution protocol	Translates an Internet address into a hardware address
AN	Auto-negotiate	Mechanism that detects the modes of two devices and automatically configures the devices to the highest common performance mode
BIOS	Basic input/output system	Collection of software routines that allows communication
BOOTP	Bootstrap protocol	Enables a diskless device to discover its own IP address
DHCP	Dynamic host configuration protocol	Allocates IP addresses dynamically
DNS	Domain name server	Program/computer that converts a domain name into its IP address
FTP	File transfer protocol	Used to transfer files across a network
HTML	Hyper text mark-up language	Used to create web pages
HTTP	Hyper text transfer protocol	Used to transmit web pages
ICMP	Internet control message protocol	Used to report errors from IP level and above
IP	Internet protocol	Mechanism for delivering packets across a network
ISP	Internet service provider	Company that links an end user to the Internet
LAN	Local area network	Group of devices that share a common communication line
NETBEUI	Network BIOS enhanced user interface protocol	Standardizes how computers on a network communicate
OSI	Open systems interconnection	Standard for how messages should be communicated across a network so that devices will consistently work with other devices
Ping	A diagnostic program	Utility that tests whether a specific IP address is accessible
RFC	Request for comments	Series of numbered Internet informational documents and standards that are widely followed by Internet software developers and others
SMTP	Simple mail transfer protocol	Used for sending and receiving email
SNMP	Simple network management protocol	Used by computers that monitor and manage network activity to communicate with one another and the computers they are monitoring
TCP	Transmission control protocol	Guarantees delivery of data
TFTP	Trivial file transfer protocol	Subset of FTP that does not require valid username and password
UDP	User datagram protocol	Found at the network layer along with the TCP protocol. UDP does not guarantee reliable, sequenced packet delivery. If data does not reach its destination, UDP does not retransmit, but TCP does.

Scope of This Application Note

This application note details the creation of a web server for an embedded device. The discussion will provide an overview of development with the MC9S12NE64 and the OpenTCP TCP/IP stack. This application note specifically addresses the following:

- MC9S12NE64 microcontroller unit (MCU)
- Axiom EVB9S12NE64 evaluation board
- OpenTCP open-source TCP/IP stack

This document was created to help familiarize first-time users of the MC9S12NE64 and OpenTCP TCP/IP stack with the development environment. This understanding will help speed initial MC9S12NE64 and OpenTCP TCP/IP stack application development. To do this, a walk-through of developing an open-source web server is provided. This also includes reviewing some basics of the OpenTCP API and its Metrowerks® CodeWarrior® project organization. Network-specific acronyms and terms used in this document are described in Table 1.

Figure 2 is a simplified diagram of the web server that will be developed. This diagram shows a PC remote host that requests a web page from an embedded device running an OpenTCP web server. The embedded device is able to serve the requested information, which includes an ActiveX component, back to the PC. In this example, the ActiveX component provides a dynamic web page interface to monitor and manipulate the embedded device. The ActiveX establishes a UDP connection with the remote host in order to exchange data on the network. Alternatively, a TCP connection could have been established for data transfers.

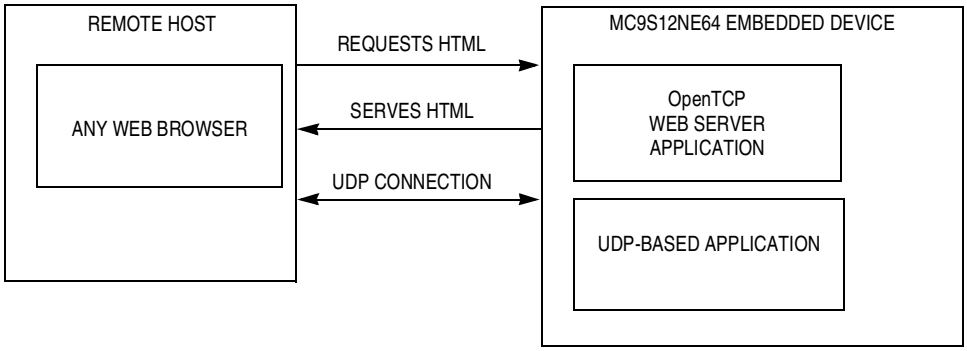

Figure 2. Web Server Example TCP/IP Stack Application

Connectivity Example Applications

Connectivity systems that use the TCP/IP stack model (see TCP/IP Stack Model Refresher), such as the example in Figure 1, can be implemented for a wide range of applications, including:

- Database data logging or queries
- Web servers for remote embedded devices
- Remote monitoring (data collection/diagnostics)
- Remote control of devices in the field
- Use of email by remote device
- Remote reprogramming of FLASH memory

TCP/IP Stack Model Refresher

The TCP/IP stack model is derived from the OSI 7-layer communications development methodology. The TCP stack model defines both TCP/IP stack software and the network interface (as shown in Figure 3). In this discussion, the network interface is Ethernet, which is implemented by the MC9S12NE64 integrated Ethernet controller and Ethernet controller device drivers.

A TCP/IP stack defines a set of protocols that allows network devices to connect to a specific device and exchange data on a network. These protocols, defined by RFC (request for comments), enable an embedded device to send email, serve web pages, transfer files, and provide other basic connectivity functions. Figure 3 is a simplified illustration of a user application working through the TCP stack model and illustrates how a TCP/IP stack and the MC9S12NE64 Ethernet controller fit into the system.

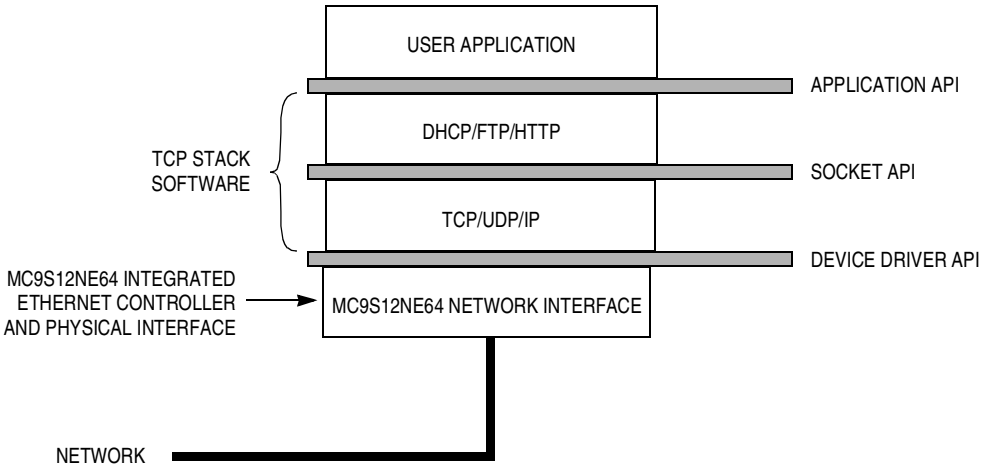

Figure 3. Block Diagram of TCP/IP Model

MC9S12NE64 MCU with Integrated Ethernet Controller

This section introduces the MC9S12NE64 MCU and provides a brief overview of the MC9S12NE64 Ethernet controller.

MC9S12NE64

The MC9S12NE64 is a 16-bit MCU based on Freescale Semiconductor's HCS12 CPU platform. It includes 8K of RAM and 64K of FLASH. The MC9S12NE64 has other standard on-chip peripherals including two asynchronous serial communications interface modules (SCIs), a serial peripheral interface (SPI), an inter-integrated circuit bus (IIC), a 4-channel/16-bit timer module (TIM), an 8-channel/10-bit analog-to-digital converter (ATD), and up to 18 pins available as keypad wake-up inputs (KWUs). In addition, an expanded bus that can be operated at 16 MHz[1] is available.

Integrated Ethernet Controller

The MC9S12NE64 introduces a new peripheral for the HCS12 CPU platform, the integrated Ethernet controller. The MC9S12NE64 integrates an Ethernet controller that includes a media access controller (MAC) and a physical transceiver (PHY) in one die with the CPU, memory, and other HCS12 standard on-chip peripherals. The MC9S12NE64 integrated Ethernet controller is compatible with IEEE 802.3 and 802.3u specifications for 10-Mbps or 100-Mbps operation, respectively.

The MC9S12NE64 can be targeted at low-throughput connectivity applications that operation at 3.3-V (±5%) external supply range. With an on-chip bandgap-based voltage regulator (VREG), the internal digital supply voltage of 2.5 V (V_{DD}) can also be generated.

A block diagram of the MC9S12NE64 is provided in Figure 4. More information on the MC9S12NE64 is available from the Freescale website: http://freescale.com/semiconductors.

1. At a 16-MHz internal bus speed, the MC9S12NE64 integrated Ethernet controller is limited to 10-Mbps operation. A 25-MHz internal bus speed is required for 100-Mbps operation. Expanded bus is available only in the 112-pin LQFP package.

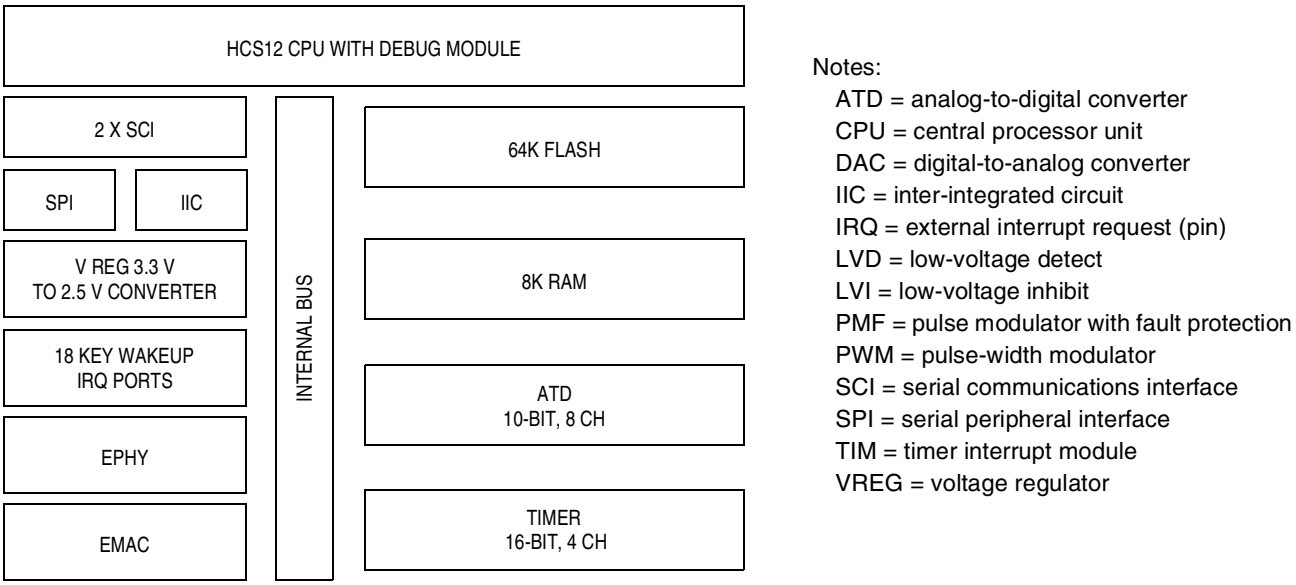

Figure 4. Block Diagram of the MC9S12NE64

Axiom Ethernet Development Board for the MC9S12NE64, EVB9S12NE64

This section describes the EVB9S12NE64 and how it must be configured for the web server demonstration.

EVB9S12NE64

Axiom Manufacturing provides the EVB9S12NE64, a fully featured development board for the MC9S12NE64. The EVB9S12NE64, shown in Figure 5, includes the following:

- MC9S12NE64 single chip Ethernet solution
- RJ45 connector with integrated Ethernet high-speed LAN magnetics isolation module
- 25-MHz crystal
- Reset button
- BDM connector
- Two RS-232C interfaces (with one configurable to an IrDA transceiver)
- Four user LEDs
- Four user buttons
- Potentiometer
- 256K external RAM (accessible via the external bus)
- Prototype area
- Access to all MC9S12NE64 pins

See the Axiom Manufacturing website, http://www.axman.com, for more information. Because the MC9S12NE64 connects directly to an Ethernet connector and high-speed LAN magnetic isolation module, the MC9S12NE64 is a true single-chip Ethernet system solution.

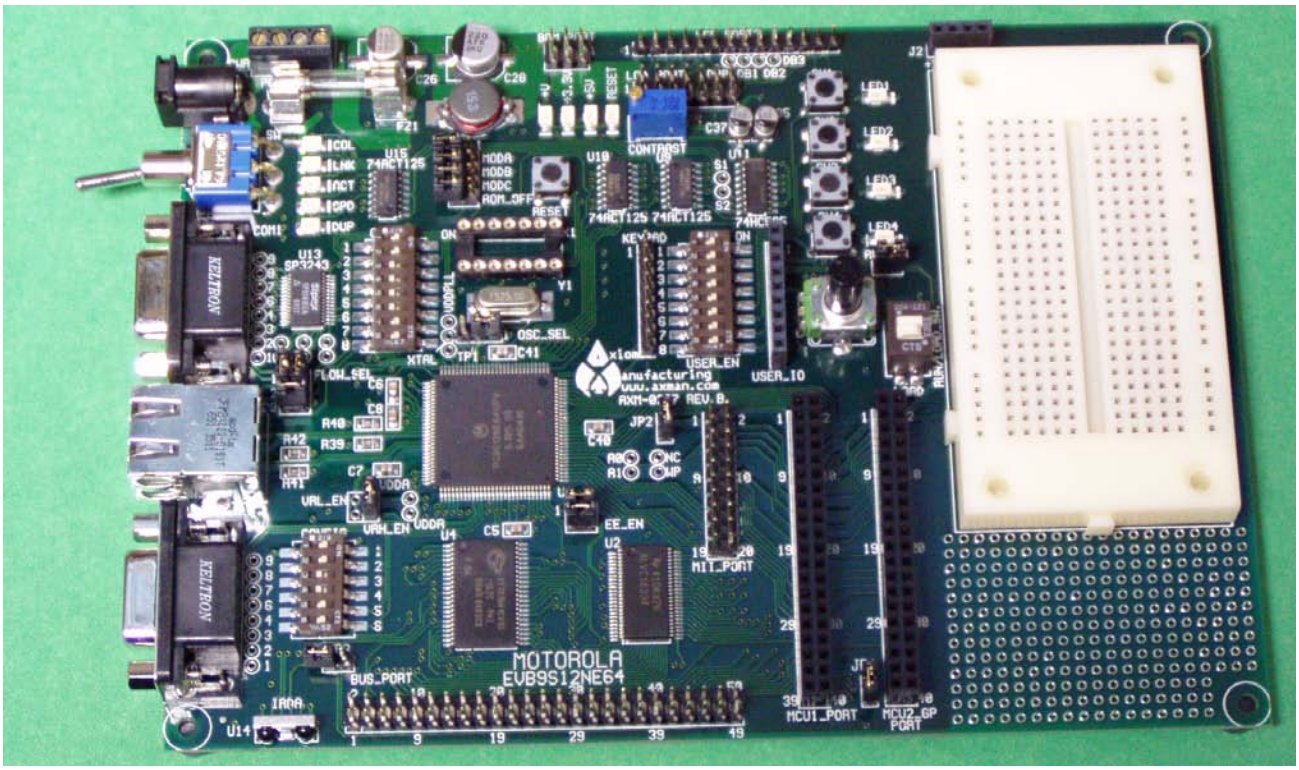

Figure 5. EVB9S12NE64 Evaluation Board from Axiom Manufacturing

EVB9S12NE64 Board Settings for Web Server Demo

The PCB version number of the EVB9S12NE64 used in this application note is revision B. For this version of EVB9S12NE64, important jumper settings for the EVB9S12NE64 are provided in Table 2. The simplified layouts of the EVB9S12NE64 top layer and board silk-screen are provided in Figure 6.

In this example, the EVB9S12NE64 is configured for normal single-chip mode (MODA=MODB=0, MODC=1). For detailed information about this evaluation board, see the EVB9S12NE64 user manual from the Axiom Manufacturing web site.

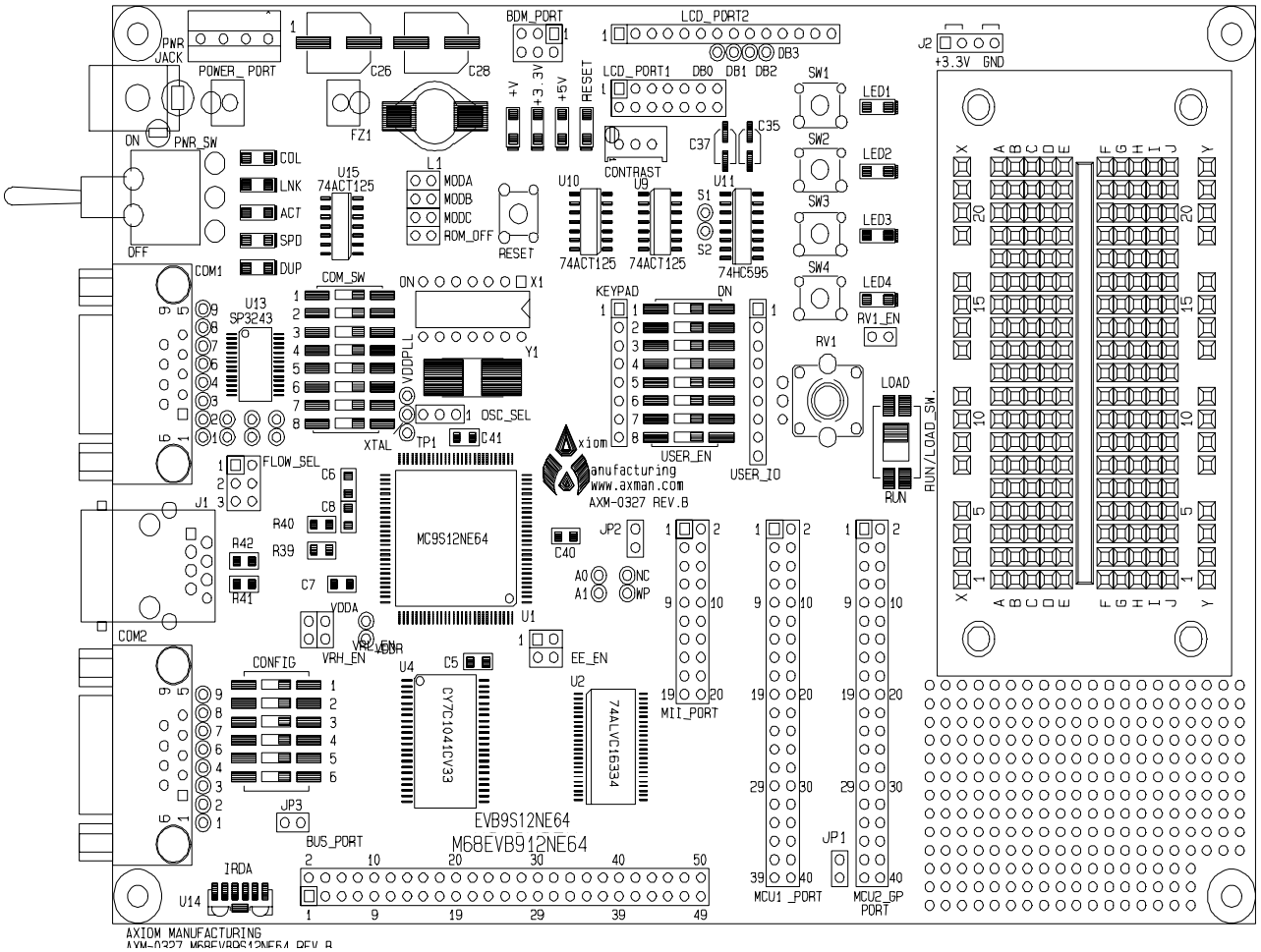

Figure 6. EVB9S12NE64 Top Layer and Silkscreen

Table 2. Settings for the EVB9S12NE64

EVB9S12NE64 Jumper/Switch	Settings
OSC_SEL, select Y1 crystal oscillator circuit	1 to 2
CONFIG switch 1–4	Off
CONFIG switch 5	Don’t care
CONFIG switch 6	On
COMM_SW 1–8	Don’t care
PWR_SW	On
USER_EN switch, RVI_EN	Don’t care
MODA, MODB, MOBC, and ROM_OFF	Off
FLOW_SEL	Don’t care
EE_EN (enable EEPROM access)	Don’t care
VRH_EN	On
JP1	Off
JP2	Off
JP3 (IRDA shutdown)	On
RUN/LOAD_SW (serial monitor)	Depends on MCU programmer interface

OpenTCP TCP/IP Stack Overview

This section includes:

- OpenTCP introduction
- Freescale Semiconductor low-level Ethernet drivers
- OpenTCP installation and project organization
- OpenTCP CodeWarrior project
- OpenTCP TCP/IP stack API

OpenTCP Introduction

OpenTCP is a robust and portable implementation of the TCP/IP and Internet application-layer protocols. Originally developed by Viola Systems (http://www.violasystems.com/index.php), OpenTCP was released under an Open Source license. It includes protocols such as ARP, IPv4, ICMP, UDP, TCP, HTTP, BOOTP, TFTP, POP3, and SMTP. A port of OpenTCP is provided for the MC9S12NE64 at SourceForge (the direct web address is http://freescaleotcp.sourceforge.net/). SourceForge is the world's largest repository of Open Source code, providing a forum for MC9S12NE64 OpenTCP developers to provide enhancements and resolve issues.

OpenTCP is a CodeWarrior compatible TCP/IP stack implementation that is tailored for 8-bit and 16-bit embedded processors. To reduce the OpenTCP code footprint in FLASH, ROM, and RAM, OpenTCP has made several TCP/IP stack design choices that deviate from TCP/IP's RFC standards while still maintaining high TCP/IP stack functionality and interoperability. These include:

- No support for IEEE 802.3 type packets
- No IP option support
- No support to handle fragmented packets
- ICMP supports only echo reply
- Ignores all TCP options
- Every TCP packet must be acknowledged with an ACK before another one can be received

Freescale Semiconductor Low-Level Ethernet Drivers

Powering the OpenTCP TCP/IP stack is a low-level Ethernet driver for the MC9S12NE64 integrated Ethernet controller. This driver is integrated within the OpenTCP TCP/IP stack source code. However, a Freescale stand-alone (without a TCP/IP stack) version of the low-level Ethernet driver is available for stand-alone development.

OpenTCP Project Installation and Organization

The OpenTCP source code is provided in a zip file. When the OpenTCP zip file is extracted, several directories are placed on the target PC. Figure 7 illustrates the OpenTCP project directories.

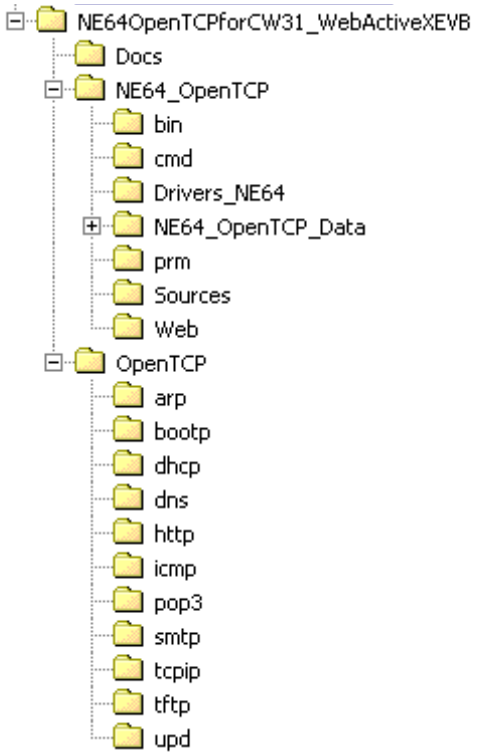

Figure 7. OpenTCP Project Directory Structure

Table 3 is a description of several sub-directory that are shown in Figure 7.

Table 3. OpenTCP Project Sub-Directory Descriptions

Sub-Directory	Description
{Project Directory}\Docs	Contains OpenTCP user manual.
{Project Directory}\NE64_OpenTCP	Contains the CodeWarrior project files for the OpenTCP project including *ne64_OpenTCP.mcp*.
{Project Directory}\OpenTCP	Contains the OpenTCP source code.
{Project Directory}\NE64_OpenTCP\bin	Contains the CodeWarrior project binaries for programming.
{Project Directory}\NE64_OpenTCP\prm	Contains the CodeWarrior project prm files for each target.
{Project Directory}\NE64_OpenTCP\Ne64_drivers	Contains the MC9S12NE64 Ethernet low-level driver source code.
{Project Directory}\NE64_OpenTCP\Sources	Contains the end application source code including main().
{Project Directory}\NE64_OpenTCP\web	Contains the end application webpage related files, C file representations of the web related files (HTML, JPG, GIF), a file to C utility, *CEncoderBeta.exe*, that converts from webpage-related files to their C representations, a batch file that can be used to automate usage of *CEncoderBeta.exe*, and an OpenTCP hash value calculator to generate a hash value from a filename, *hashcalbeta.exe*.

More information on *ne64_OpenTCP.mcp* is provided in the following sections. Figure 8 shows *ne64_OpenTCP.mcp* opened in the CodeWarrior IDE (integrated development environment).

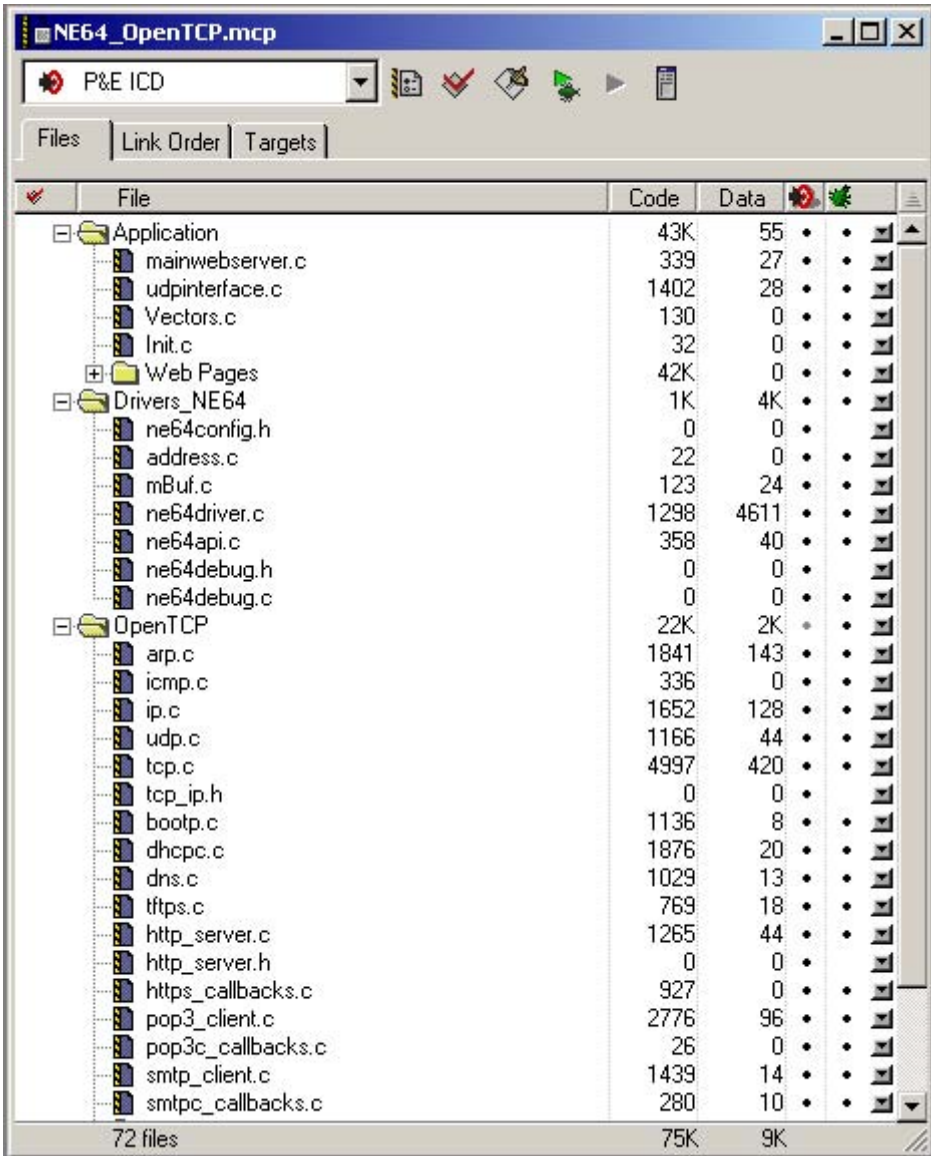

Figure 8. CodeWarrior IDE with OpenTCP Projects Open

OpenTCP CodeWarrior Projects

This section details the files in *ne64_OpenTCP.mcp*. The specific files that are discussed are the files that will likely require modification in order to develop the simple web server described by this application note.

Table 4. Files in ne64_OpenTCP.mcp

Category	Filename	Description
OpenTCP	debug.h	OpenTCP file for debug options
	arp.c	OpenTCP ARP implementation
	icmp.c	OpenTCP ICMP implementation
	ip.c	OpenTCP IP protocol implementation
	udp.c	OpenTCP UDP implementation
	tcp.c	OpenTCP TCP implementation
	dhcpc.c	OpenTCP DHCP client implementation
	FileSys.c	Contains a file system array for HTTP
	http_server.c	OpenTCP HTTP server implementation
	https_callbacks.c	OpenTCP HTTP callback implementation
	dns.c	OpenTCP DNS client implementation
	pop3_client.c	OpenTCP POP3 client implementation
	pop3c_callbacks.c	OpenTCP POP3 callback functions
	smtp_client.c	OpenTCP SMTP client implementation
	smtpc_callbacks.c	OpenTCP SMTP callback implementation
	tftps.c	OpenTCP TFTP server implementation
	bootp.c	OpenTCP BOOTP client implementation
MC9S12NE64 Driver	ne64config.h	Used to configure low-level EMAC and EPHY options
	address.c	Used to configure the MAC address and IP address
	ne64driver.c	Low-level initialization code
	mBuf.c	Ethernet buffer descriptions
	ne64api.c	TCP/IP stack interface functions
	ne64debug.c	Provides a debug interface
Application	Vectors.c	Provides an ISR array
	webserver.c	Contains the main()
	Init.c	Provides the application initialization code
	udpinterface.c	Provides demo application code

OpenTCP TCP/IP Stack API

Table 5 provides a more detailed look at OpenTCP by reviewing some basic API OpenTCP functions including a brief description for each function. For complete documentation of the OpenTCP API, please reference the OpenTCP user guide.

Table 5. Selected OpenTCP API Functions

	Function	Description
UDP API Functions	udp_init();	Initializes UDP socket pool to get everything into a known state at startup and should be called before any other UDP function.
	INT8 udp_releasesocket (INT8)	Releases a given socket.
	INT8 udp_open (INT8, UINT16)	Opens a given UDP socket for communication.
	INT8 udp_getsocket (UINT8, INT32(*)(INT8, UINT8, UINT32, UINT16, UINT16, UINT16), UINT8)	Allocates a free socket in UDP socket pool.
	INT16 udp_send (INT8, UINT32, UINT16, UINT8 *, UINT16, UINT16)	Sends data to remote host using given UDP socket.
TCP API Functions	tcp_init();	Initializes TCP socket pool to get everything into a known state. It should be called before any other TCP function. Timers are also allocated for each socket and everything is brought to a predefined state.
	tcp_poll();	Checks all TCP sockets and performs various actions if timeouts occur. What kind of action is performed is defined by the state of the TCP socket.
	INT8 tcp_listen (INT8, UINT16)	Sets TCP socket to listen on a given port.
	INT8 tcp_connect (INT8, UINT32, UINT16, UINT16)	Initializes connection establishment towards remote IP&port.
	INT16 tcp_send (INT8, UINT8 *, UINT16, UINT16)	Sends user data over TCP using given TCP socket.
	INT8 tcp_getsocket (UINT8, UINT8, UINT16, INT32(*)(INT8, UINT8, UINT32, UINT32))	Allocates a free socket in TCP socket pool.
HTTP API Functions	https_init();	Initializes HTTP server variables; it should be called before the HTTP server application is used to set operating parameters.
	https_run();	Is main thread of HTTP server program and should be called periodically from main loop.
Other Support Functions	EtherInit();	Configures and initializes the MC9S12NE64 EMAC and EPHY.
	arp_init();	Initializes ARP cache table so that ARP allocates and initializes a timer for its use.

Preparing for OpenTCP TCP/IP Stack Development

This section describes how to prepare for web server development with OpenTCP software. The following topics are included:

- Development environment and tools
- Connecting the evaluation board to a development PC
- Configuring the MAC hardware and IP addresses
- Configuring TCP/IP protocol in Microsoft Windows® operating system

Development Environment and Tools

The OpenTCP TCP/IP stack software can be modified and compiled with the CodeWarrior environment. Below are specific details about the development environment and tools used to develop the web server described in this application note.

- Microsoft Windows 2000 with Microsoft Internet Explorer 5.5 or later
- Microsoft FrontPage® 2000 for web page development
- CodeWarrior HCS12 tool version 2 (with MC9S12NE64 patch) or later
- P&E BDM MultiLink® pod, category 5 (cat5) crossover cable, and optional DB9 serial cable (as described in the next section)

Connecting the Evaluation Board to a Development PC

Figure 9 shows the basic connection of a PC running CodeWarrior software to the EVB9S12NE64. For development, a PC must connect to the target board with the following:

- BDM MultiLink pod (BDM) — Provides a link to the embedded device and provides an interface to program and debug the software on the MC9S12NE64 MCU. On the EVB9S12NE64 evaluation board, the BDM connector is labeled BDM_PORT. Alternatively, instead of BDM, the target could be programmed and debugged using a serial cable along with the HCS12 serial monitor.
- Crossover cat5 cable (XCAT5) — Required to form a local, isolated network between the PC and the target. With an Ethernet link, the network application can be tested and debugged on a network. Alternatively, a straight-through cable with a hub can be used.
- DB9 serial cable (COMM) — The OpenTCP program has a debug mode that sends real-time messages about stack activity through one of the MC9S12NE64 SCI ports. By using HyperTerminal[1] and a serial cable, these debug messages can be captured.

1. To access HyperTerminal in Windows systems, click Start> Programs> Accessories> Communication> HyperTerminal.

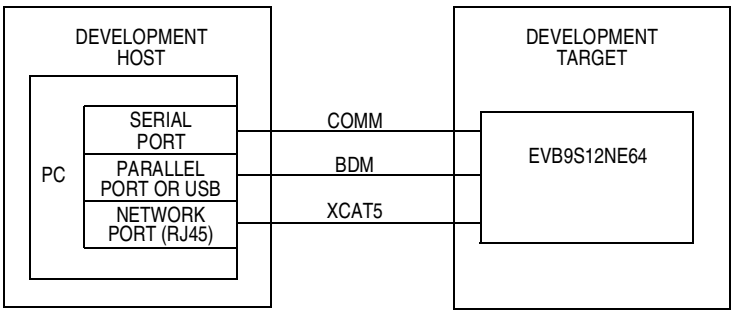

Figure 9. Connecting the Evaluation Board to a PC

This step provides an easier interface for debugging the application than connecting directly to a real network (because the development target is isolated). To make the application compatible with a real network, changes should be required to only the MAC hardware and IP addresses.

Configuring the MAC Hardware and IP Addresses

After the web server is complete, it can be configured to operate on a real network by changing the MAC hardware and IP addresses to be compatible with the real network. A brief discussion of MAC hardware and IP addresses is provided in the following sections.

Configuring the MAC Hardware Address

The MAC hardware address is a 48-bit number. Each network device must have a unique MAC hardware address. MAC hardware address groups are assigned to organizations by the IEEE EtherType Field Registration Authority.

A valid MAC hardware address for the EVB9S12NE64 should be assigned by the developer. This address is used by the datalink layer, which is implemented by the MC9S12NE64 integrated Ethernet controller and the low-level drivers. This can be configured in *address.c*. If the device is not connected to a real network, a random MAC hardware address can be used as long as it is not connected on a network that has a device with the same 48-bit MAC hardware address.

Configuring the IP Addresses

IP addresses are assigned by a network administrator or a dynamic host configuration protocol (DHCP) server. These addresses are used by the IP layer of the OpenTCP TCP/IP stack. If the IP addresses are not correctly configured, the embedded device will not communicate over the network connection—even if an Ethernet connection can be made.

When developing an application off a real network and on a developer's PC, the developer is the network administrator. The developer must create a local network between the PC and the target board. A network consists of nodes that are on the same network subnet. To set up the subnet for the demo, the developer must use compatible IP address settings between the developer's PC and the target board.

When setting up IP addresses, it is preferable to configure or use the development target and development host manually with non-routable IP addresses (i.e., 10.x.x.x, 90.0.0.x, 172.16.x.x through 172.32.x.x, or 192.168.x.x).

IP address settings for this demo:

- All devices are configured with an IP address in the range 192.168.2.1 to 192.168.2.3
- OpenTCP code is programmed with an IP address of:
  - 192.168.2.1, for the development host (PC)
  - 192.168.2.3, for the development target (embedded device)

NOTE

These IP settings and others must be reflected in the Windows network settings.

When the application is developed and ready for a real network, the IP address settings must be configured to be compatible with the real network on which the node will reside. Recall, for the OpenTCP code, IP addresses and MAC hardware addresses are configured in *addess.c*. In this web server example, a static IP address is used. For a real network, recall that the IP address should be assigned by the network administrator. Optionally, the DHCP capability of OpenTCP software can be used. With DHCP, a DHCP server will automatically assign a leased IP address to the embedded device.

Configuring TCP/IP Protocol in Windows

To set up the IP address for the development host in Windows, the IP address network settings of the development host must be accessed in its operating system. For recent Windows releases, these settings are located in the control panel. In the control panel, select network settings. A typical network settings dialog box is shown in Figure 10.

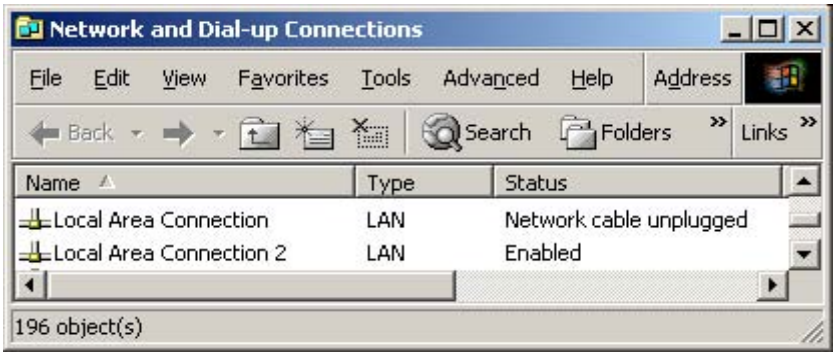

Figure 10. Network Settings Dialog Box

The network settings window shows all devices that can be used to form network connections. Figure 10 shows two network devices defined for the PC. Network devices can also include modems.

The network settings window also shows the status of the network device. This status indicates whether the network device has an Ethernet connection. Having an Ethernet connection does **not** necessarily mean other Windows network settings for that device are set correctly (see Debugging Networks).

To access the IP address setting via the network settings dialog box, you must select the desired network device and configure its properties. In the properties dialog box, select the TCP/IP protocol network component for the TCP/IP adapter (see Figure 11) and click Properties.

In the Internet Protocol (TCP/IP) Properties dialog box, manually enter a subnet mask and specific IP address. Recall that the IP address used for the development host in this example is 192.168.2.1.

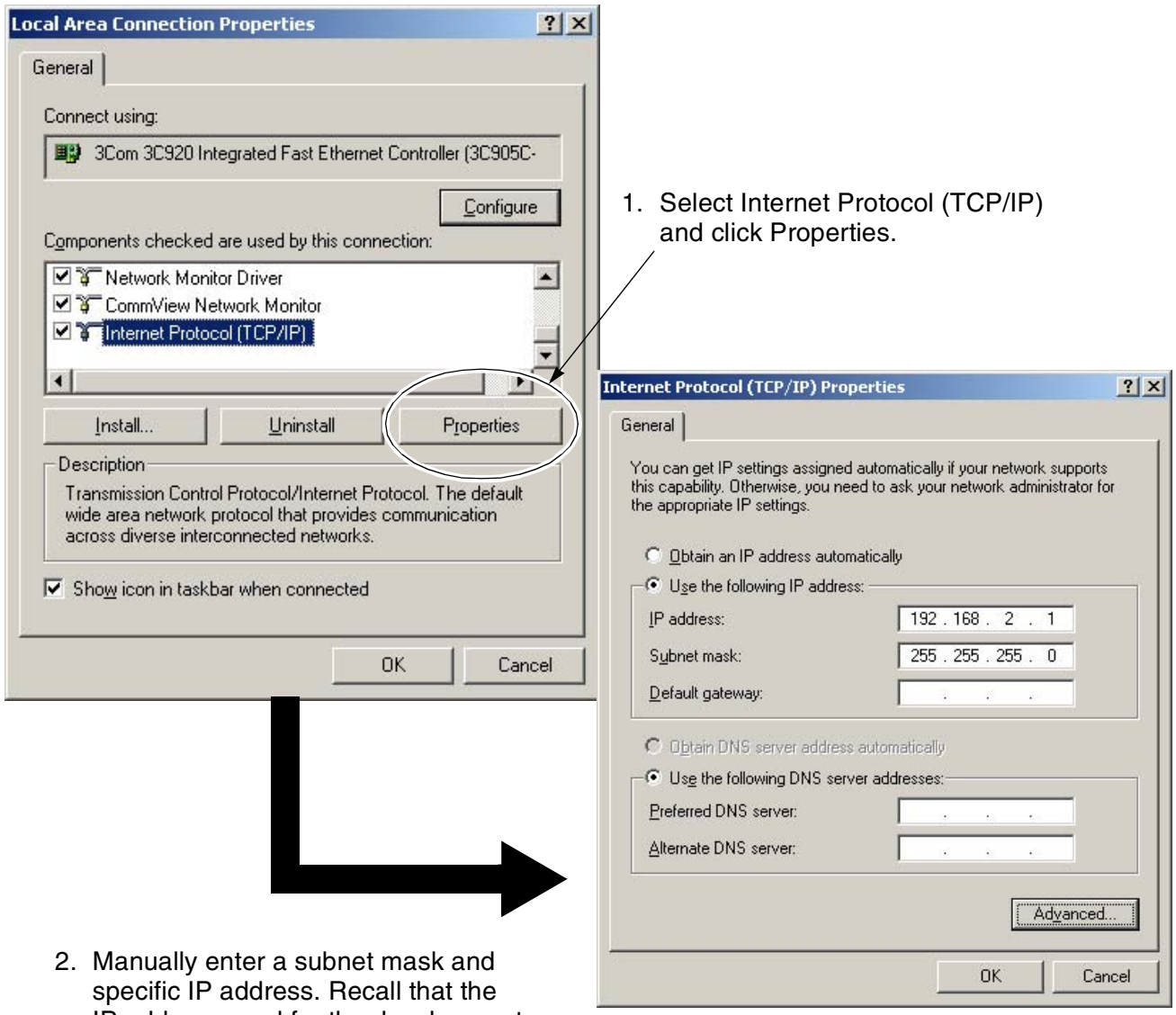

Figure 11. Opening Internet Protocol (TCP/IP) Properties Dialog Box

Debugging Networks

Issues

Issues with network connectivity are typically due to an error in the network setup and configuration of either the network or the remote devices. Three main network connectivity issues and their possible solutions are described in this section:

- Ethernet connection not established
- Network connection cannot be established
- Network connection is established at the IP layer with Ping, but the devices are not communicating

Ethernet Connection Not Established

This connectivity issue means that the Ethernet transceiver on either the target or the host (or both) cannot create a low-level link. This problem can be caused by:

- Cat5 cable damaged or unplugged
- Cat5 cross-over cable required, but a straight-through cable is used
- Ethernet transceiver loss of power
- Ethernet transceiver issue at startup
- Mismatched Windows LAN card settings

Ethernet transceiver issues at startup occur if the embedded device was not initialized correctly by the program. First, visually verify that the devices are physically connected. On the PC and the target evaluation board, the link and speed LEDs should be active.

If the physical connection is visually verified and the problem still exists, check the status of the link in the network settings dialog box as shown in Figure 10. Also, if the network is configured correctly, Windows may show the status of the link on the task bar as shown in Figure 12. The task bar shows the link as “Network Cable Unplugged.”

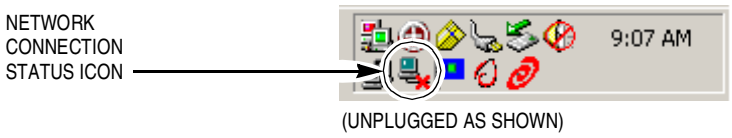

Figure 12. Task Bar Showing Link Status

Network Connection Cannot be Established

After the device establishes an Ethernet connection, the network settings may still require adjustments. The most common problems are:

- Devices are unreachable
- Network connection is misdirected

In these two cases, the network IP address, network port information, network components, network protocols, and server type must be reviewed.

This section deals with the issue of the network higher level protocols not establishing a connection, such as an Internet protocol (TCP/IP) connection. One way this can be checked is with Ping. Ping is actually an IPv4 ICMP (internet control management protocol) echo request that is defined by the TCP/IP stack protocol. Because Ping functionality is included in OpenTCP and Windows, it can be used to debug the network connection. If the command confirms a valid connection to a remote device by replying to the ICMP echo request, the network is configured correctly. If, however, the Windows command does not confirm a network connection to the remote device, the network is not configured correctly.

The key step to resolve this type of network bug is to determine how the network is designed and how the remote device must be configured to accept a connection. Remote devices must be compliant with the network's design structure and protocols. When the network is set up and configured correctly, the devices will connect. This problem is usually associated with incorrect and incompatible IP address settings (see Configuring the IP Addresses).

Network Connection is Established at the IP Layer with Ping, but the Devices are Not Talking

This problem is usually difficult to debug. There may be a conflict with other protocols settings. Other possible causes can be a firewall, proxy server settings, duplex mismatch, or invalid server settings. With an understanding of the network design and its connection capabilities, network restrictions, and underlying communication protocols (for example, TCP/IP and NETBEUI), a user can configure the network and the remote devices to ensure connectivity. This issue may require assistance from a system administrator to resolve.

Network Protocol Analyzer Tools

A network protocol analyzer is a powerful and useful tool for network debugging. The network protocol analyzer enables more visibility of packet traffic on the network connection. A network protocol analyzer is used to monitor the connectivity of the Internet or a local area network (LAN).

The tool is capable of non-intrusively attaching itself and monitoring a dial-up or Ethernet connection. The network protocol analyzer can be an in-house, commercial, or downloadable freeware software package. A network protocol analyzer can be implemented in hardware also.

The overriding feature of the network protocol analyzer is its ability to capture, analyze, and decode network packets. The network protocol analyzer must be capable of determining the communication protocol of the network data packets. In addition, the program must be able to display a list of network connections, the IP addresses of the connections, the data direction, and the network data port information. The network protocol analyzer provides the detailed network information required to debug a network.

Overview of a Web Server Developed Using OpenTCP TCP/IP Stack

This section provides an overview of the web server and source code developed with OpenTCP. A graphical overview of the web server is shown in Figure 2.

This web server is developed for distribution with the EVB9S12NE64 as a binary file, *P&E_ICD.abs*. *P&E_ICD.abs* is located in the in the bin directory in a typical CodeWarrior project. Figure 13 shows *demo.htm* in a web browser. *demo.htm* is one of the four pages stored on the web server. The other three pages on the web server are static. *demo.htm*, however, provides dynamic web page functionality using an embedded ActiveX control, *NE64DemoNOPIC.ocx*. *NE64DemoNOPIC.ocx* is stored on the web server as part of the *NE64DemoNOPIC.cab* file and is developed to fit in the MC9S12NE64 internal FLASH memory along with the other web page data.

The ActiveX control for the example was built with Microsoft Studio Visual Basic 6, but this is not a requirement. Another ActiveX development IDE could have be used. In fact, instead of using an ActiveX control, a Java applet or other web technology could be used to add the dynamic web page functionality. This discussion does not include a step-by-step guide of how to develop an ActiveX component with Visual Basic, but does provide some high-level ActiveX development considerations.

The ActiveX control, in this example, contains several basic Windows component objects, which are itemized in Table 6. Each object adds to the overall ActiveX functionality as shown in Table 6.

Table 6. Windows Component Objects

Component	Quantity	Description
Winsock	1	Provides a TCP or UDP connection
Timers	2	Provide periodic timer events
ToggleButton	6	Provides a two-state button
ProgressBar	1	Provides a graphical view of the current value within a range
CommandButton	3	Provides a user command button
TextBox	2	Provides a user data input or display field
Shape	4	Provide a round graphical object
Label	12	Provides GUI labels for headings

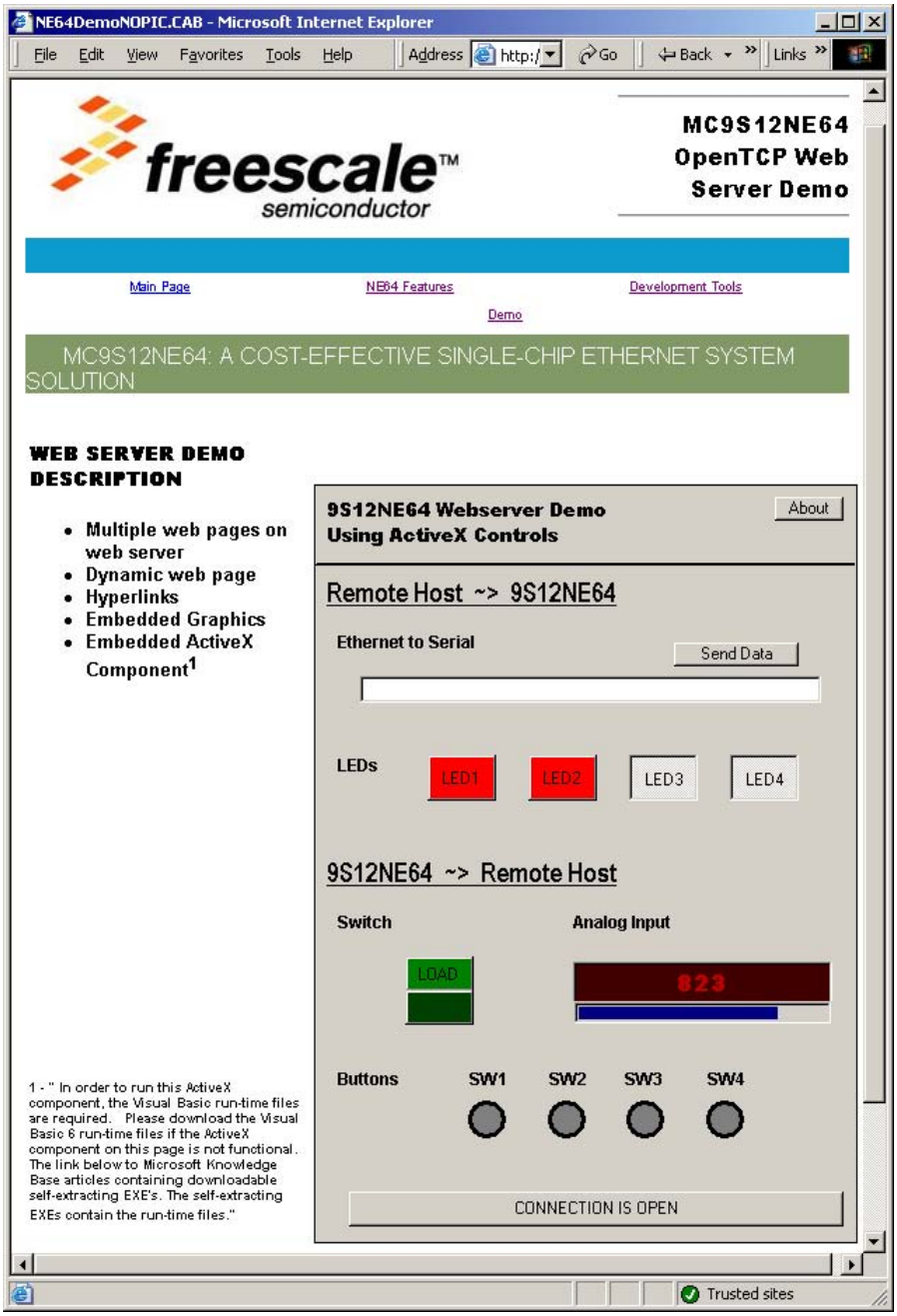

Figure 13. OpenTCP Web Server

The remainder of this section will overview some of the files and source code that were modified during development of the OpenTCP web server demo. Acronyms and terms used in this section are defined in Table 1.

address.c

Before coding the web server application code, it is recommended that the basic MC9S12NE64 hardware and software options are configured. These configuration options include:

- MAC hardware address
- IP address
- Auto-negotiation
- EMAC options
- EPHY options
- Ethernet buffer size

Although most of the options are controlled by *ne64config.h*, *address.c* provides an interface to configure the web server MAC address and IP address with the hard_addr and prot_addr variables, respectively. (See the code excerpt below.)

address.c:

```
/*****************************************************************************
 *
 *                      (c) Freescale  Inc. 2004 All rights reserved
 *
 * File Name     : address.c
 *
 * Description    : This file contains definition of hardware and protocol (IP)
 *                  addresses of the device
 *
 * Version : 2.1
 * Date    : 02/04/04
 *
 *****************************************************************************/

#include "MOTTYPES.h"

const tU08 hard_addr [6] = { 0x01, 0x23, 0x45, 0x56, 0x78, 0x9a };

const tU08 prot_addr [4] = { 192, 168,   2,   3 };
const tU08 netw_mask [4] = { 255, 255, 255,   0 };
const tU08 dfgw_addr [4] = { 192, 168,   2,   1 };
const tU08 brcs_addr [4] = { 192, 168,   2, 255 };
```

ne64config.h

An excerpt of the source code for *ne64config.h* is provided below. *ne64config.h* allows the user to configure initialization options for the MC9S12NE64 EMAC and EPHY. The code excerpt below shows a partial list of the configurations a user may desire to modify. These configurations are itemized below along with a brief description of each:

- AUTO_NEG — This option enables the auto-negotiation mode if asserted. If not asserted, the developer must manually set up the speed and duplex using the full_duplex and speed_100 variables.
- HALF10, FULL10, HALF100, & FULL100 — These set the auto-negotiation advertisements and should be set to 1 if that mode is to be advertised in auto-negotiation.
- SPEED100 — if AUTO_NEG = 0 — This manually sets the device link speed.
- FULL_DUPLEX — if AUTO_NEG = 0 — This manually sets the device link duplex.
- ETYPE_PET, ETYPE_EMW, ETYPE_IPV6, ETYPE_ARP, ETYPE_IPV4, ETYPE_IEEE, and ETYPE_ALL — These set the EMAC ethertype filtering modes.
- BRODC_REJ, CON_MULTIC, and PROM_MODE — The sets the EMAC MAC address filtering modes.
- BUFMAP — BUFMAP configures the RAM amount that is allocated for the Ethernet buffer. For the MC9S12NE64, 8K of RAM is available and shared between user RAM and the EMAC Ethernet buffer space. A more detailed description of BUFMAP is provide in the *P&E_ICD_linker.prm* section.

Excerpt from ne64config.h:

```
.
.
.
//=============================================================
//LINK SPEED/DUPLEX CONTROL
//=============================================================
//Configure for manual or auto_neg configuration
#define AUTO_NEG     1    /**< 1 - enable AUTO_NEG / 0 - disable AUTO_NEG */

#if  AUTO_NEG

//define what I advertise in auto negotiation
#define HALF100   1
#define FULL100   1
#define HALF10    1
#define FULL10    1
.
.
.
#else                     //AUTO_NEG
#define SPEED100    1      /**< 1 - enable 100 MBps / 0 - enable 10 MBps */
#define FULL_DUPLEX 0      /**< 1 - enable full duplex / 0 - disable */

#endif                    //AUTO_NEG
```

```
//============================================================
//Buffer Configuration
//============================================================
#define BUFMAP 4
.
.
.
//============================================================
//EMAC FILTERING CONTROL
//============================================================
//Address Filtering; RXMODE setting: PAUSE frame supported, Accept Unique, Brodcast, MultiCast
#define         BRODC_REJ       0
#define   CON_MULTIC            0
#define   PROM_MODE             0
//Ethertype Control
#define ETYPE_PET   0
#define ETYPE_EMW   0
#define ETYPE_IPV6  0
#define ETYPE_ARP   0
#define ETYPE_IPV4  0
#define ETYPE_IEEE  0
#define ETYPE_ALL   1

//Programmable Ethertype
#define ETYPE_PRG  0     /**< Enter Value if ETYPE_PET is set for filter target */

//Receive maximum frame length
#define RX_MAX_FL        1536    /**< Receive maximum frame length */
#define DELETE_BFRAMES    0     /**< set to 1 to delete packets larger than RX_MAX_FL */
.
.
.
```

P&E_ICD_linker.prm

An excerpt of the source code for *P&E_ICD_linker.prm* is provided below. *P&E_ICD_linker.prm* must be modified so that it is compatible with *ne64config.h*'s BUFMAP configuration settings. BUFMAP configures the Ethernet buffer space which consist of three buffer (one transmit buffer and two receive buffers) that are each designed to only hold one Ethernet frame at any given time. The BUFMAP field is a component of the BUFCFG register.

The RAM section shows the remainder available system RAM for the user application. This RAM for the user application is the difference between the total system RAM (8K) and the amount of system RAM allocated for the Ethernet buffer.

Table 7 itemizes the remainder of RAM available for the user application based on the value of BUFMAP. Because the maximum size of an Ethernet frame is approximately 1.5K, a setting of BUFMAP = 4 would allow each of the MC9S12NE64's three buffers to hold one frame of the maximum allowable size.

Table 7. Memory Allocation

BUFMAP	Individual Buffer Size (bytes)	Total Size of EMAC Ethernet buffer space	Remainder RAM for User Application
0	128	384	7.625K
1	256	768	7.25K
2	512	1536	6.5K
3	1K	3072	5K
4	1.5K	4608	3.5K

Setting for BUFMAP less than 4 are provided to:

- Maximize user RAM
- Provide a filtering mechanism base on Ethernet packet size

The setting of BUFMAP is a system/network design decision. If the devices on your network should only accept Ethernet packets of a certain size limit, BUFMAP should be configured accordingly.

Setting BUFMAP to create a buffer size limit reduces the burden on CPU from processing larger undesirable packets on the network. If a receive frame exceeds the receive buffer size, the frame is not accepted and both receive complete flag or the receive error flag are **not** set. No CPU bandwidth is used because the EMAC state machine does all the packet filtering. This is a powerful filtering mechanism.

P&E_ICD_linker.prm also defines the placement of the sections. It is noteworthy to discuss the MyConstSegPage1 placement. MyConstSegPage1 refers to a web server file array for the ActiveX component, NE64DemoNOPIC_file, which is contained in *NE64DemoNOPIC.c*. Because *NE64DemoNOPIC.c* is a constant array, by default, it would be stored in ROM_VAR which causes a FLASH allocation problem for this project. Creating MyConstSegPage1 and grouping the NE64DemoNOPIC_file data with DEFAULT_ROM resolves the FLASH allocation problem. More information on FLASH allocation can be found within the CodeWarrior user manual.

Excerpt from P&E_ICD_linker.prm:

```
NAMES
END

SECTIONS

//    RAM       = READ_WRITE 0x2180 TO 0x3FFE;     /* BUFMAP = 0 (128 byte) */
//    RAM       = READ_WRITE 0x2300 TO 0x3FFE;     /* BUFMAP = 1 (256 byte) */
//    RAM       = READ_WRITE 0x2600 TO 0x3FFE;     /* BUFMAP = 2 (512 byte) */
//    RAM      = READ_WRITE 0x2C00 TO 0x3FFE;     /* BUFMAP = 3 (1K) less then means no DHCP*/
    RAM       = READ_WRITE 0x3200 TO 0x3FFE;     /* BUFMAP = 4 (1.5K) */

    /* unbanked FLASH ROM */
ROM_4000 = READ_ONLY  0x4000 TO 0x7FFF;
ROM_C000 = READ_ONLY  0xC000 TO 0xFEFF;

/* banked FLASH ROM */
SECURITY = READ_ONLY  0xFF00 TO 0xFF0F;
ROM_FF10 = READ_ONLY  0xFF10 TO 0xFF7F;
```

```
PAGE_3C  = READ_ONLY  0x3C8000 TO 0x3CBFFF;
PAGE_3D  = READ_ONLY  0x3D8000 TO 0x3DBFFF;
END

PLACEMENT
    _PRESTART,                  /* Used in HIWARE format: jump to _Startup at the code start */
    STARTUP,                      /* startup data structures */
    ROM_VAR,                      /* constant variables */
        STRINGS,                      /* string literals */
    NON_BANKED,                   /* runtime routines which must not be banked */
    COPY                          /* copy down information: how to initialize variables */
                                  INTO  ROM_4000,ROM_C000;
        MyConstSegPage1,
    DEFAULT_ROM                   INTO   PAGE_3C,PAGE_3D;
    DEFAULT_RAM                   INTO  RAM;
END

STACKTOP 0x3FFF
```

Demo.htm

The HTML source code for *demo.htm* is provided in this section. HTML web page development can be assisted with web tools, such as FrontPage by Microsoft, but it is not required. A simple text editor could be used, for example. Web pages can contain standard and advanced HTML components including, but not limited to:

- Frames
- Tables
- Forms
- Javascript or VBscript
- Applets, ActiveX, etc.

When the web page development is completed, the next step is to convert them to their equivalent C files representation using OpenTCP's CEncoderBeta utility. In this case, the C files for *demo.htm* are *demo.c* and *demo.h*. Both of these files must be added to OpenTCP CodeWarrior project and detailed in the OpenTCP file system (see the FileSys.c section).

The source code below is standard HTML. The portion of the code that embeds the ActiveX component on the web page is the following:

```
<object ID="UserControlPanel" CLASSID="CLSID:956FC15B-5D58-4F1D-95CC-242020FA9175"
CODEBASE="NE64DemoNOPIC.CAB#version=1,0,0,0">
<param name="_ExtentX" value="10266">
<param name="_ExtentY" value="14340">
</object>
```

This object shows the ActiveX component is provided in a cabinet file, *NE64DemoNOPIC.CAB*. Cabinet files are used to package ActiveX controls for faster downloading over the internet by reducing the ActiveX file size footprint using data-compression technology. The cabinet file also includes a "inf" file to automate the ActiveX component file installation and registration in the client machine's registry.

Web Server demo.htm HTML Source Code:

```
<html>

<head>
<meta http-equiv="Content-Language" content="en-us">
<meta name="GENERATOR" content="Microsoft FrontPage 5.0">
<meta name="ProgId" content="FrontPage.Editor.Document">
<title>NE64DemoNOPIC.CAB</title>
</head>

<body bgcolor="#FFFFFF">

<table border="1" cellpadding="0" cellspacing="0" style="border-collapse: collapse;
border-width: 0" bordercolor="#111111" width="100%" id="AutoNumber4">
<tr>
<td width="72%" style="border-style: none; border-width: medium"><img border="0"
src="freescale.jpg" align="left" width="313" height="117"> </td>
<td width="28%" style="border-style: none; border-width: medium"><hr>
<p align="right" style="margin-top: 0; margin-bottom: 0"><font size="4" face="Arial
Black">MC9S12NE64 OpenTCP Web Server Demo</font></p>
<hr></td>
</tr>
<tr>
<td width="100%" style="border-style: none; border-width: medium" colspan="2"
bgcolor="#0C99CC"><font size="1" face="sans-serif" color="#FFFFFF"><img border="0"
src="space.gif" width="27" height="22"></font> </td>
</tr>
<tr>
<td width="100%" style="border-style: none; border-width: medium" colspan="2">
<p style="margin-bottom: 8" align="center"><font size="1" face="sans-serif"><a
href="index.htm">Main Page</a><img border="0" src="space.gif" width="125" height="13"><a
href="features.htm">NE64 Features</a><img border="0" src="space.gif" width="125"
height="13"><a href="devtools.htm">Development Tools</a><img border="0" src="space.gif"
width="125" height="13"><a href="demo.htm">Demo</a></font><font size="1" face="sans-serif"
color="#FFFFFF"><img border="0" src="space.gif" width="27" height="17"></font> </td>
</tr>
<tr>
<td width="100%" style="border-style: none; border-width: medium" colspan="2"
bgcolor="#809966"><font size="1" face="sans-serif" color="#FFFFFF"><img border="0"
src="space.gif" width="27" height="21"></font><font face="sans-serif" size="4"
color="#FFFFFF">MC9S12NE64: <span style="text-transform: uppercase">A cost-effective
single-chip Ethernet system solution </span></font></td>
</tr>
</table>
<table border="1" cellpadding="0" cellspacing="0" style="border-collapse: collapse;
border-width: 0" bordercolor="#111111" width="101%" id="AutoNumber5">
<tr>
<td width="3%" style="border-style: none; border-width: medium">  </td>
<td width="43%" style="border-style: none; border-width: medium" valign="top"> <p><font
face="Arial Black"><b>WEB SERVER DEMO DESCRIPTION</b></font></p>
<ul>
<li><font face="sans-serif"><b>Multiple web pages on web server</b></font></li>
<li><font face="sans-serif"><b>Dynamic web page</b></font></li>
<li><b><font face="sans-serif">Hyperlinks</font></b></li>
<li><b><font face="sans-serif">Embedded Graphics</font></b></li>
<li><font face="sans-serif"><b>Embedded ActiveX Component<sup>1</sup></b></font></li>
```

```
</ul>
<p><font size="1" face="sans-serif" color="#FFFFFF"><img border="0" src="space.gif" width="27"
height="244"></font> </p>
<p><font face="sans-serif" size="1"><b>1 - &quot; In order to run this ActiveX component, the
Visual Basic run-time files are required.   Please download the Visual Basic 6
run-time files if the ActiveX component on this page is not functional. The link below to
Microsoft Knowledge Base articles containing downloadable self-extracting EXE's. The
self-extracting EXEs contain the run-time files.&quot;</b></font> </p>
<p><b><a HREF="http://support.microsoft.com/default.aspx?scid=fh;en-us;vbruntime&product=vbb"
style="text-decoration: none"><font COLOR="#0000ff" face="sans-serif"
size="1">http://support.microsoft.com/default.aspx?scid=fh;en-us;vbruntime&amp;product=vbb</f
ont></a></b> </p>
<p>  </td>
<td width="7%" style="border-style: none; border-width: medium">  </td>
<td width="48%" style="border-style: none; border-width: medium">
<object ID="UserControlPanel" CLASSID="CLSID:956FC15B-5D58-4F1D-95CC-242020FA9175"
CODEBASE="NE64DemoNOPIC.CAB#version=1,0,0,0">
<param name="_ExtentX" value="10266">
<param name="_ExtentY" value="14340">
</object>
</td>
</tr>
</table>
<table border="1" cellpadding="0" cellspacing="0" style="border-collapse: collapse;
line-height: 100%; text-align: center; border-width: 0; margin-top: 10; margin-bottom: 10"
bordercolor="#111111" width="100%" id="AutoNumber6" bgcolor="#0C99CC">
<tr>
<td width="100%" style="border-style: none; border-width: medium" bgcolor="#0C99CC"
align="center">
<p style="margin-top: 0; margin-bottom: 0" align="right"><font size="1" face="sans-serif"
color="#FFFFFF"><img border="0" src="space.gif" width="27" height="22"></font><font
face="sans-serif" size="1">© Freescale Semiconductor, Inc. 2004. All Rights
Reserved</font><font size="1" face="sans-serif" color="#FFFFFF"><img border="0"
src="space.gif" width="15" height="21"></font> </td>
</tr>
</table>

</body>

</html>
```

FileSys.c

FileSys.c is used by OpenTCP as a file system array reference for HTML and other files used in the web server. When a client remote host makes a request to the web server for a file, OpenTCP accesses *FileSys.c* to determine whether the file is available, so it is important that *FileSys.c* contains the correct information. *FileSys.c* requires that the header files provided by *CEncoderBeta.exe* are listed in the include section for *FileSys.c* as shown in the source code.

FileSys.c defines an array of the web server files. Each file entry is defined by the following structure, which includes a hash value of the original web server file name, a pointer to the C representation of the web server file, and its file size. For *demo.htm*, the pointer to the C representation of the web server file is found in *demo.c* (demo_file); the length of the *demo.c* date is provided in *demo.h* (DEMO_FILE_LEN).

```
typedef struct TFileEntry
{
unsigned char hash;         // hash value for this entry file name
const unsigned char* file_start_address; //start of file
unsigned short        file_length;
} TFileEntry;
```

The hash value for *demo.htm* can be easily calculated. The hash calculation formula is provided in the https_calculatehash (UINT32 len) function in the *http_server.c* file. A simple utility is provided with this application note (in AN2863SW.zip from freescale.com) to provide the hash calculation, *hashcalculator.exe*. This hash calculator is provided without any guarantees.

Table 8 provides a brief description of each file referenced by the web server.

Table 8. Files Referenced by the Web Server

Filename	Hash Value	Description
index.htm	115	Static web page
devetools.htm	139	Static web page
features.htm	242	Static web page
demo.htm	160	Dynamic web page
freescale.jpg	193	Logo graphic
space.jpg	120	Graphic
NE64DemoNDPIC.cab	247	Compressed ActiveX file

Excerpt from FileSys.c:

```
/*****************************************************************************
 *
 *                      (c) Freescale  Inc. 2004 All rights reserved
 *
 * File Name     : FileSys.c
 * Description   :
 *
 * Version : 1.0
 * Date    : Mar/05/2004
 *
 *
 *****************************************************************************/
 #include "FileSys.h"

/********* Include Web server files here *************/
#include "index.h"
#include "NE64DemoNOPIC.h"
#include "demo.h"
#include "devtools.h"
#include "features.h"
#include "freescale.h"
#include "space.h"
/***************************************************/

const TFileEntry FAT [] = {
     {  115,    index_file,      INDEX_FILE_LEN      },
     {  247,    NE64DemoNOPIC_file,      NE64DEMONOPIC_FILE_LEN  },
     {  160,    demo_file,      DEMO_FILE_LEN  },
     {  139,    devtools_file,      DEVTOOLS_FILE_LEN  },
     {  242,    features_file,      FEATURES_FILE_LEN  },
     {  193,    freescale_file,      FREESCALE_FILE_LEN  },
     {  120,    space_file,      SPACE_FILE_LEN  },
     {  0,      (unsigned char *)0,  0  }
};
```

mainwebserver.c

mainwebserver.c contains main() for the user application. In main(), the MC9S12NE64 MCU and its integrated Ethernet controller are configured and enabled. main() also initializes the OpenTCP TCP/IP stack then waits and serves a web page on request.

mainwebserver.c Source Code:

```
/*****************************************************************************
 *
 *       Copyright (C) 2003 Freescale Semiconductor, Inc.
 *           and 2000-2002 Viola Systems Ltd.
 *       All Rights Reserved
 *
 *****************************************************************************/

#include "debug.h"
```

```
#include "datatypes.h"
#include "timers.h"
#include "system.h"
#include "ethernet.h"
#include "arp.h"
#include "ip.h"
#include "tcp_ip.h"

#include "http_server.h"
#include "smtp_client.h"

#include "ne64driver.h"
#include "ne64api.h"
#include "mBuf.h"
#include "ne64config.h"
#include "udp_demo.h"

#include "address.h"

#include "MC9S12NE64.h"
/* Including used modules for compiling procedure */

/* Network Interface definition. Must be somewhere so why not here? :-)*/
struct netif localmachine;

extern void RTI_Enable (void);
extern void porth_isr_handler (void);

extern  tU08    gotlink;

#if USE_SWLED
tU16 LEDcounter=0;
#endif


//=============================================================
tU08 OldSwitchValue=255;
tU16 Pot=0;
tU16 OldPot=1050;
tU08 OldB1=255;
tU08 OldB2=255;
tU08 OldB3=255;
tU08 OldB4=255;

//=============================================================
#pragma CODE_SEG NON_BANKED
interrupt void PortHInterrupt (void)
{
  porth_isr_handler();
}
#pragma CODE_SEG DEFAULT


//=============================================================
//Initialize ATD
//=============================================================
void ATD_init(void)
```

```
{
    ATDCTL2 =  ATDCTL2_ADPU_MASK | ATDCTL2_AFFC_MASK;
    ATDCTL3_S1C = 8;    // 8 ch seq.
    ATDCTL3_FIFO = 0;    // no FIFO
    ATDCTL3_FRZ = 3;     // Freeze immediately in BDM
    ATDCTL4 =  ATDCTL4_PRS2_MASK |ATDCTL4_PRS1_MASK | ATDCTL4_PRS0_MASK;
    ATDCTL4 = ATDCTL4 & ~ATDCTL4_SRES8_MASK; //10 bit
    ATDCTL5 = ATDCTL5_SCAN_MASK;
}


//============================================================
// Initialize Port for LEDs, Switch, and Buttons
//============================================================
void demoinit(void)
 {
     //LEDS
    DDRG_DDRG0 = 1;                                 //1:output
    DDRG_DDRG1 = 1;                                 //1:output
    DDRG_DDRG2 = 1;                                 //1:output
    DDRG_DDRG3 = 1;                                 //1:output
                PTG_PTG0 = 1;//turn off
                PTG_PTG1 = 1;//turn off
                PTG_PTG2 = 1;//turn off
                PTG_PTG3 = 1;//turn off

    //SWITCH (RUN/LOAD) 0:input
    DDRG_DDRG4 = 0;

    //BUTTON2
    DDRH_DDRH4 = 0;
    PIEH_PIEH4 = 1;      //PIEH4 Interrupt Enable

    DDRH_DDRH5 = 0;        //0:input
    DDRH_DDRH6 = 0;        //0:input


 }


//============================================================
/* main */
//============================================================
void main(void)
{
  INT16 len;

  /* System clock initialization */
  CLKSEL=0;
  CLKSEL_PLLSEL = 0;                    /* Select clock source from XTAL */
  PLLCTL_PLLON = 0;                     /* Disable the PLL */
  SYNR = 0;                            /* Set the multiplier register */
  REFDV = 0;                           /* Set the divider register */
  PLLCTL = 192;
  PLLCTL_PLLON = 1;                     /* Enable the PLL */
  while(!CRGFLG_LOCK);                  /* Wait */
  CLKSEL_PLLSEL = 1;                    /* Select clock source from PLL */
```

```
  INTCR_IRQEN = 0;                     /* Disable the IRQ interrupt. IRQ interrupt is enabled
after CPU reset by default. */


        /* initialize processor-dependant stuff (I/O ports, timers...).
         * Most important things to do in this function as far as the TCP/IP
         * stack concerns:
         *  - initializing some timer so it executes decrement_timers
         *      on every 10ms (TODO: Throw out this dependency from several files
         *      so that frequency can be adjusted more freely!!!)
         *  - not mess too much with ports allocated for Ethernet controller
         */
    init();
    demoinit();
    ATD_init();
        /* Set our network information. This is for static configuration.
     * if using BOOTP or DHCP this will be a bit different.
     */
        /* IP address */
          localmachine.localip = *((UINT32 *)ip_address);
        /* Default gateway */
        localmachine.defgw   = *((UINT32 *)ip_gateway);
        /* Subnet mask */
        localmachine.netmask = *((UINT32 *)ip_netmask);

        /* Ethernet (MAC) address */
        localmachine.localHW[0] = hard_addr[0];
        localmachine.localHW[1] = hard_addr[1];
        localmachine.localHW[2] = hard_addr[2];
        localmachine.localHW[3] = hard_addr[3];
        localmachine.localHW[4] = hard_addr[4];
        localmachine.localHW[5] = hard_addr[5];


        /* Init system services*/
        timer_pool_init();

    /* Initialize all buffer descriptors */
        mBufInit ();

        /*interrupts can be enabled AFTER timer pool has been initialized */

  /* Initialize all network layers*/
  EtherInit();

  //Initialize required network protocols
  arp_init();
  (void)udp_init();

        udp_demo_init();
  (void)tcp_init();
        (void)https_init ();

        //Enable RTI
        RTI_Enable ();
```

```
        /* main loop */
        DEBUGOUT(">>>>>>>>>>Entering to MAIN LOOP>>>>>>>>>>\n\r");
        for (;;)
        {
#if USE_SWLED
      UseSWLedRun();
#endif
          if (gotlink) {

                /* take care of watchdog stuff */
                /* Try to receive Ethernet Frame*/
                if( NETWORK_CHECK_IF_RECEIVED() == TRUE ) {
                          switch( received_frame.protocol)
                            {
                                case PROTOCOL_ARP:
                                                  process_arp (&received_frame);
                                break;
                                case PROTOCOL_IP:
                                                len = process_ip_in(&received_frame);
                                                if(len < 0)
                                                break;
                                                switch (received_ip_packet.protocol)
                                                  {
                                                case IP_ICMP:
                                                process_icmp_in (&received_ip_packet, len);
                                                  break;
                                                case IP_UDP:
                  process_udp_in (&received_ip_packet,len);
                  break;
                                                case IP_TCP:
                                                process_tcp_in (&received_ip_packet, len);
                                                break;
                                                default:
                                                break;
                                                }
                                break;
                                  default:
                                  break;
                            }
                          /* discard received frame */
                          NETWORK_RECEIVE_END();
                }
    arp_manage();

                /* Application main loops */
                /* TCP/IP stack Periodic tasks here... */
    udp_demo_run();
          tcp_poll();
    https_run ();

          }
          else {
                PTG_PTG0 = 1;//turn off LED1
                PTG_PTG1 = 1;//turn off LED2
          }
        }
}
```

OpenTCP Project Configuration to Optimize the Stack Solution

When developing a web server, there are several strategies to ensure that the code size of the solution does not exceed the available resources:

- Minimize or off-load web page content
- Use only required network protocols
- Set buffers to appropriate values

Minimize or Off-Load Web Page Content

A fully featured web page for an application uses valuable FLASH and RAM resources. Before implementation, it is important to understand the resources that the application will require and balance them with the web server features. Each web page graphic, for example, can easily require 6 to 8 Kbytes of FLASH. The following techniques help minimize or off-load web server content and still provide the connectivity functionality for the application:

- Use an HTML file compressor — HTML compressors remove unnecessary white space characters such as carriage returns, line feeds, spaces, and tabs; this can reduce the size of a web page by as much as 50%. Compressed code requires less space on the web server and client system. More importantly, compressed HTML files download faster.
- Compress any ActiveX controls using a cabinet file as described in the Demo.htm section. Java is compressed using “jar” files.
- Place ActiveX controls or other content on another web server — the CODEBASE attribute in the Demo.htm section shows that the ActiveX cabinet file is on the web server. Alternatively, this code can be stored on another, larger web server that is reachable by the network in the end application. The alternative syntax for CODEBASE to point to another location is CODEBASE="http://www.largewebserver.com/offloadedforNE64.cab#Version=1,0,0,1". This off-loads the web server content from the embedded device to another web server.
- Use a standalone UDP (or TCP) application that is loaded on the client machine instead of building a web server to provide the remote access. This off-loads the web server content from the embedded device to the client machine. This is not possible in all cases, but this approach should be considered. An example UDP application that can replace the web server while providing similar functionality is provided as an example (see Figure 14).

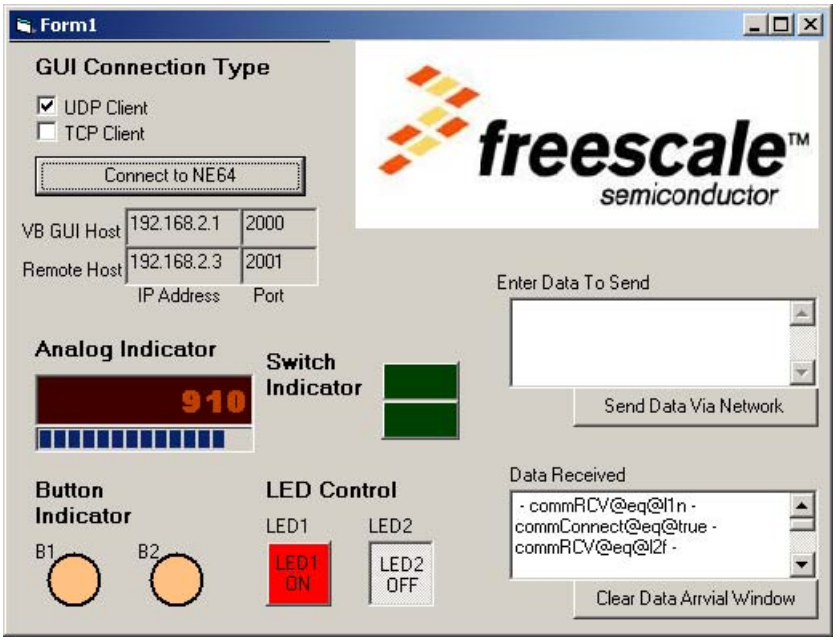

Figure 14. Example of UDP Application Replacing the Web Server

- Off-load web page data to an external memory device interfaced to the MC9S12NE64 via the expanded memory or using one of the MC9S12NE64 serial interfaces. Softec (www.softecmicro.com) provides a OpenTCP demo that demonstrates using external memory via the SPI. The Softec example stores all web page content on the external device. An example of the Softec web content is provided in Figure 15.

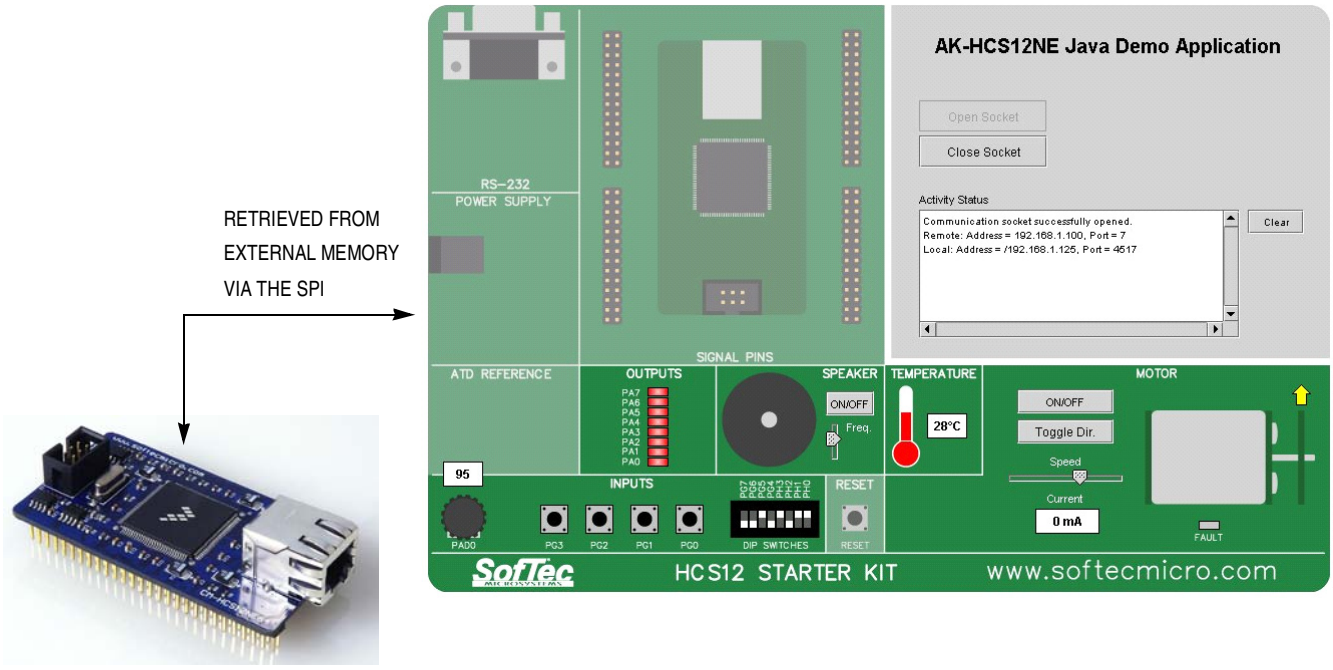

Figure 15. Softec Web Content

Use Only Required Network Protocols

For resource-constrained TCP/IP stack implementations, such as implementing a TCP/IP stack on an 8-/16-bit embedded system, it is not always best to implement the complete set of networking protocols. Figure 16 illustrates a simplified or partial TCP/IP stack implementation. This stack uses only the UDP protocol and the user applications.

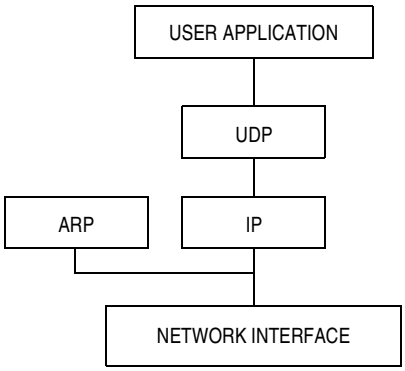

Figure 16. Partial Stack TCP/IP Stack Implementation

The major disadvantage of TCP/IP stack implementations that are customized to a specific application is that they are not complete TCP/IP stack implementations. So, if changes to the TCP stack functionality are required after product deployment, making updates would require recompiling the TCP stack code to include the missing components and reprogramming the device in the field.

OpenTCP Zero Copy Functionality

Zero copy refers to the TCP/IP stack ability to send and receive from the Ethernet buffers' stack without copying data to user RAM. Not making a copy of the data saves user RAM space and improves performance. Performance is improved because, in most cases, the data can be completely processed in much less of time that would be required to make a copy of the data between the Ethernet buffer RAM and web RAM.

Set Buffers to Appropriate Values

RAM for Tx and Rx Ethernet buffers should be balanced with user application RAM. If large Ethernet buffers are not required for the user application, set the Ethernet buffer values (in the BUFMAP register) so that the user application uses only the necessary RAM resources. The Tx and Rx Ethernet buffers' size is controlled in *ne64config.h*.

Conclusion

Combining the MC9S12NE64 and the OpenTCP stack software provides a single-chip Ethernet system solution that is low cost and easy to use. The OpenTCP web server described in this application note is only one of the many network applications possible with the MC9S12NE64.The MC9S12NE64 provides developers with the means to add Ethernet functionality to everyday applications and/or design innovative, network-enabled applications.

NOTE

With the exception of mask set errata documents, if any other Freescale document contains information that conflicts with the information in the device data sheet, the data sheet should be considered to have the most current and correct data.

Although specific methods and tools are used to develop and debug this demo, Freescale Semiconductor does not recommend or endorse any particular methodology, tool, or vendor. These methods and tools are provided only to describe the generic principles and features that may be required for development of a networked device.

How to Reach Us:

USA/Europe/Locations not listed:
Freescale Semiconductor Literature Distribution
P.O. Box 5405, Denver, Colorado 80217
1-800-521-6274 or 480-768-2130

Japan:
Freescale Semiconductor Japan Ltd.
SPS, Technical Information Center
3-20-1, Minami-Azabu
Minato-ku
Tokyo 106-8573, Japan
81-3-3440-3569

Asia/Pacific:
Freescale Semiconductor H.K. Ltd.
2 Dai King Street
Tai Po Industrial Estate
Tai Po, N.T. Hong Kong
852-26668334

Learn More:
For more information about Freescale Semiconductor products, please visit
http://www.freescale.com

AN2836
Rev. 0, 9/2004

freescale™
semiconductor