Freescale Semiconductor
Application Note

AN2700
Rev. 0.3, 9/2004

Basic Web Server Development with MC9S12NE64 and CMX-MicroNet™ TCP/IP Stack

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.

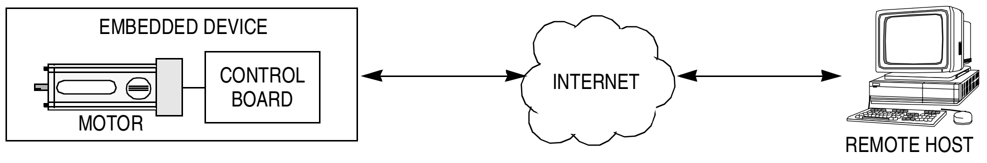


Figure 1. Embedded Device on Internet

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 basic web server for an embedded device. The application note is a follow-up to AN2624/D: *Basic Web Server Development with the CMX-MicroNet™ TCP/IP Stack*. CMX-MicroNet development with the MC9S12NE64 is very similar to CMX-MicroNet development MC9S12E128 Ethernet reference design. The discussion will provide an overview of development with the MC9S12NE64 and the CMX-MicroNet TCP/IP stack. This application note specifically addresses the following:

- MC9S12NE64 microcontroller unit (MCU)
- Axiom EVB9S12NE64 evaluation board
- CMX-MicroNet TCP/IP stack

This document was created to help familiarize first-time users of the MC9S12NE64 and CMX-MicroNet TCP/IP stack with the development environment. This understanding will help speed initial MC9S12NE64 and CMX-MicroNet TCP/IP stack application development. To do this, a walk-through of developing a simple web server is provided. This also includes reviewing some basics of the CMX-MicroNet 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 on a CMX-MicroNet web server. The embedded device is able to serve the requested information back to the PC. The remote host can also GET and POST information to the embedded device.

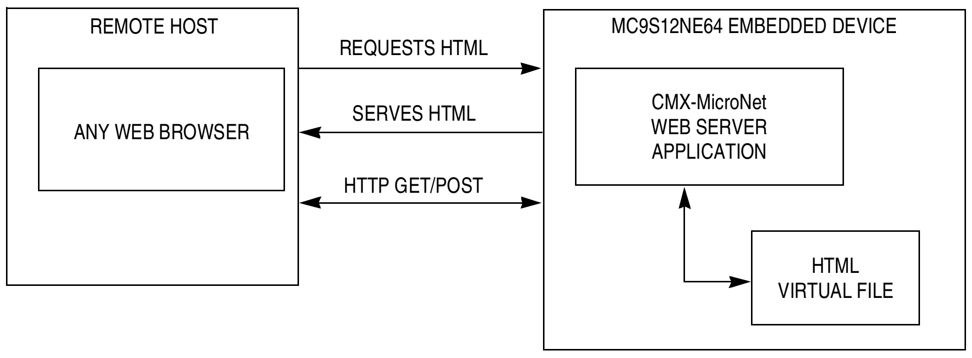


Figure 2. Web Server Example TCP/IP Stack Application

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.

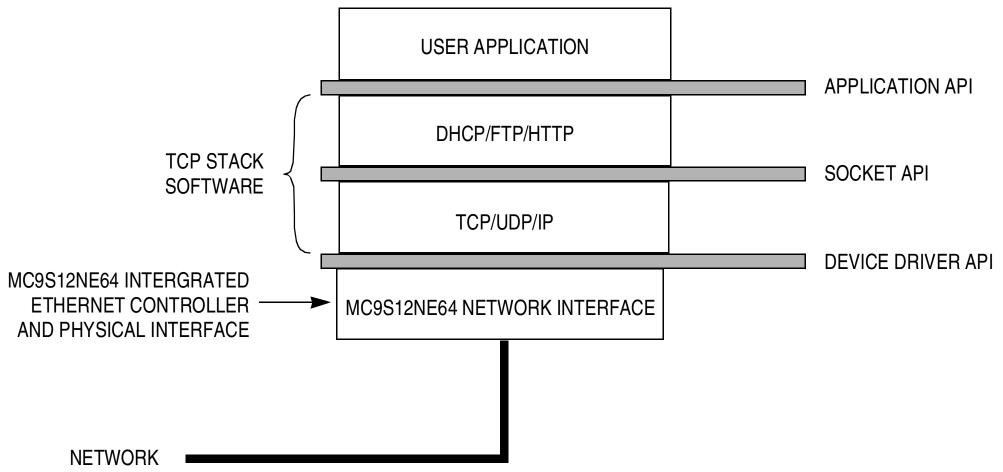


Figure 3. Block Diagram of TCP/IP Model

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. In the 80-pin TQFP-EP package, 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.

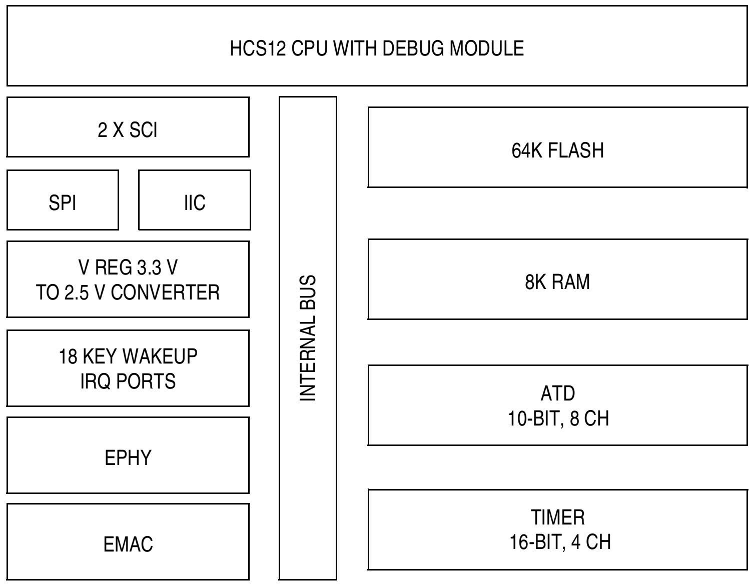
Integrated Ethernet Controller

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 require operation from a 3.15-V to 3.45-V 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 Semiconductor website: http://freescale.com.

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.



Notes:

ATD	= analog-to-digital converter
CPU	= central processor unit
DAC	= digital-to-analog converter
IIC	= inter-integrated circuit
IRQ	= external interrupt request (pin)
LVD	= low-voltage detect
LVI	= low-voltage inhibit
PMF	= pulse modulator with fault protection
PWM	= pulse-width modulator
SCI	= serial communications interface
SPI	= serial peripheral interface
TIM	= timer interrupt module
VREG	= voltage regulator

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.

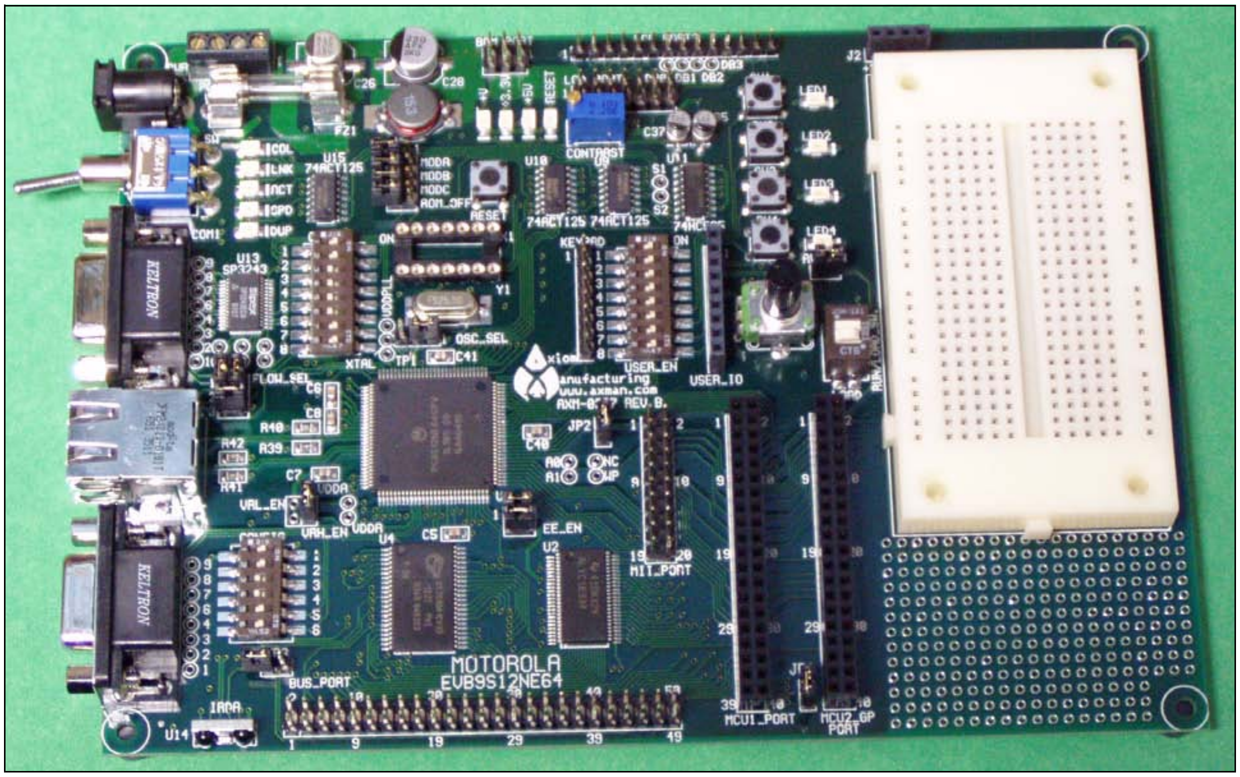


Figure 5. EVB9S12NE64 Evaluation Board from Axiom Manufacturing

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 silkscreen are provided in Figure 6.

The EVB9S12NE64 must be configured for normal single-chip mode (MODA=MODB=0, MODC=1) for the CMX-MicroNet stack operation. For detailed information about this evaluation board, see the EVB9S12NE64 user manual from the Axiom Manufacturing web site.

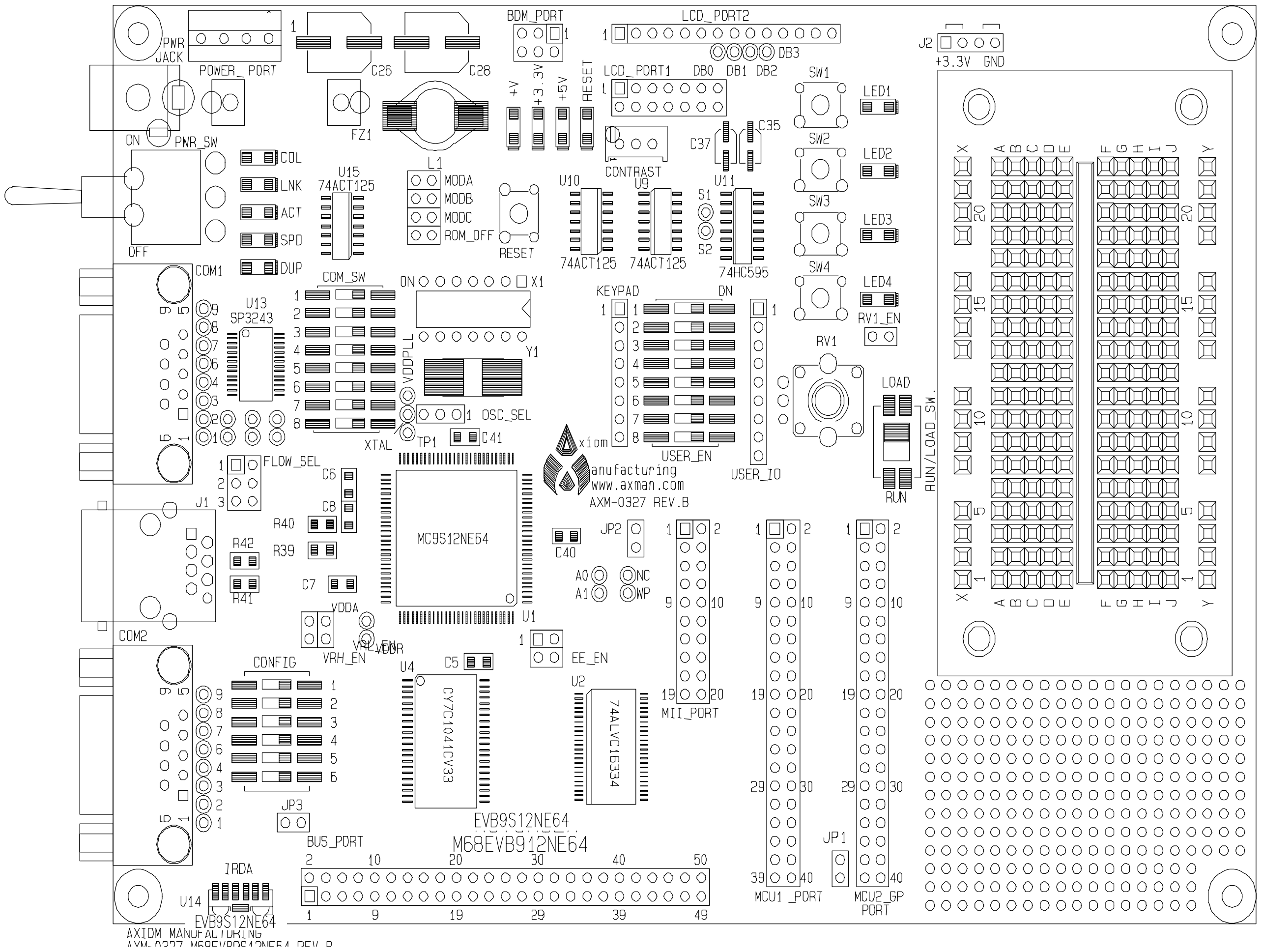


Figure 6. EVB9S12NE64 Top Layer and Silkscreen

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)	Don’t care

CMX Stack Overview

This section includes:

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

CMX Introduction

CMX-MicroNet is a TCP/IP stack implementation that is tailored for 8-bit and 16-bit embedded processors. Other CMX features include:

- Compatible with CodeWarrior tools
- Works with MCUs with low RAM/ROM resources
- Written entirely in standard C code
- Allows web pages to contain CGI calls
- Allows sending email
- Can serve Java applets
- Runs stand-alone or with a real-time operating system (RTOS)
- Supports as many as 16 sockets (mixed or matched with TCP or UDP)

To reduce the CMX-MicroNet code footprint in FLASH, ROM, and RAM, CMX-MicroNet has made several TCP/IP stack design choices that deviate from TCP/IP's RFC standards while still maintaining high TCP/IP stack functionality. These include:

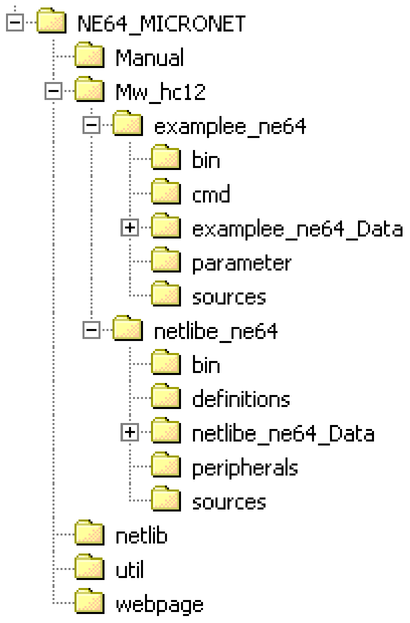
- No support for IEEE 802.3 type packets
- No IP option support
- No support to handle fragmented packets
- ICMP supports only echo reply
- Ignores all TCP options
- TCP respects other side's window, but uses a fixed window itself
- Every TCP packet must be acknowledged with an ACK before another one can be received

Freescale Semiconductor Low-Level Ethernet Drivers

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

CMX-MicroNet Project Installation and Organization

CMX delivers CMX-MicroNet source code as an installation file. After this file is installed, several directories are placed on the target PC. Figure 7 illustrates the CMX-MicroNet project directories that are installed. The CMX-MicroNet main project directory is {Project Directory}\NE64_MICRONET, where {Project Directory} is the installation directory.



```
NE64_MICRONET
    Manual
    Mw_hc12
        examplee_ne64
            bin
            cmd
            examplee_ne64_Data
            parameter
            sources
        netlibe_ne64
            bin
            definitions
            netlibe_ne64_Data
            peripherals
            sources
    netlib
    util
    webpage
```

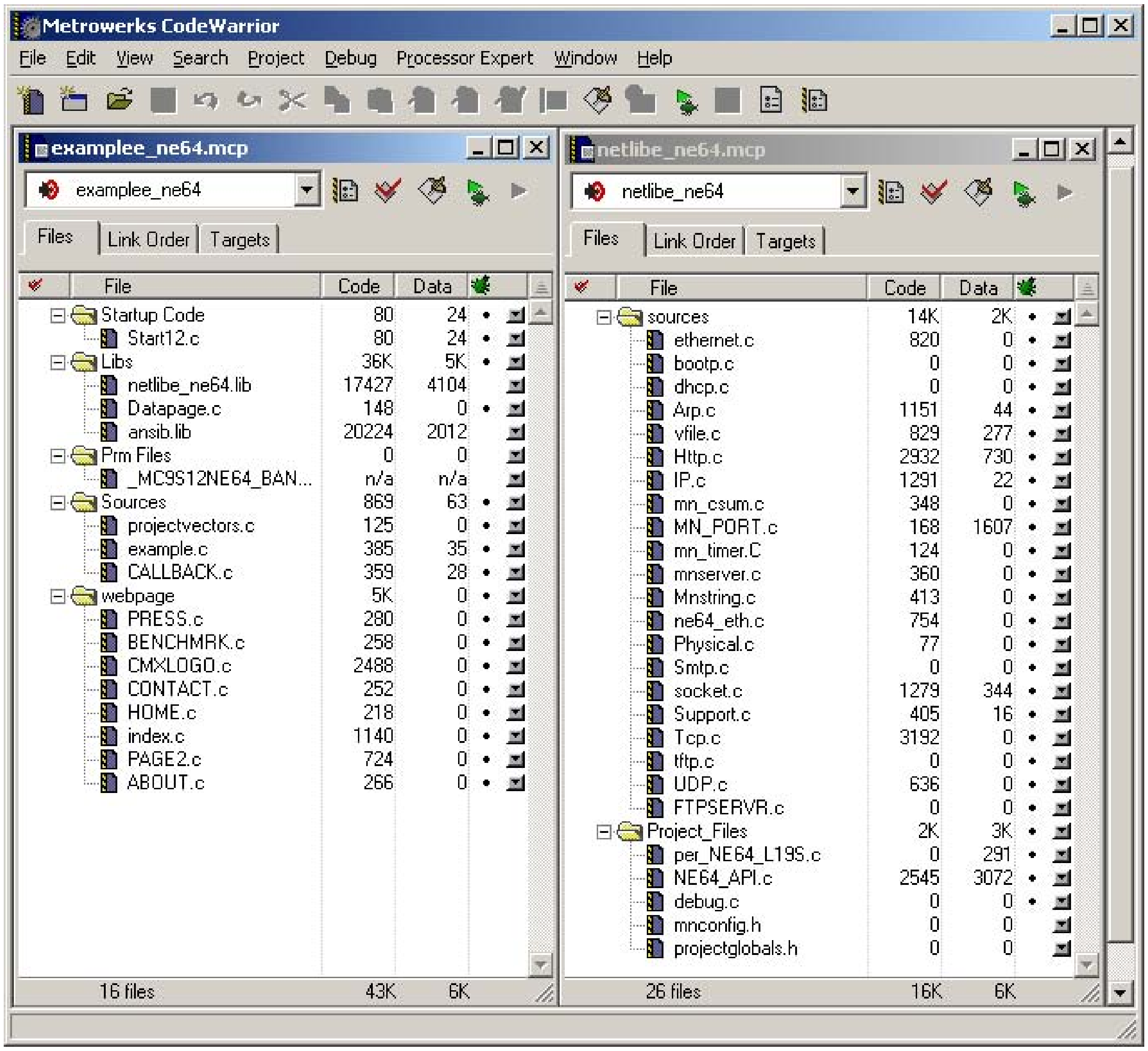
Figure 7. CMX-MicroNet Project Directory Structure

Table 3 is a description of each sub-directory in the CMX-MicroNet main project directory.

Table 3. CMX-MicroNet Project Sub-Directory Descriptions

Sub-Directory	Description
{Project Directory}\NE64_MICRONET\Manual	Contains CMX documentation and user manual.
{Project Directory}\NE64_MICRONET\netlib	Contains the source files for the CMX-MicroNet API.
{Project Directory}\NE64_MICRONET\Util	Contains the **html2C.exe** utility that can be used to convert HTML, JPG, GIF, and other files into equivalent C files for CMX-MicroNet to use for the embedded web server. (For example, if you have a web page called *main.html*, running *html2C.exe* with *main.html* as the argument will create files *main.c* and *main.h*. In this example, *main.c* and *main.h* should be added to the CMX-MicroNet CodeWarrior project.)
{Project Directory}\NE64_MICRONET\Webpage	Contains the HTML, JPG, GIF, and other files developed for the embedded web server.
{Project Directory}\NE64_MICRONET\Mw_hc12	Contains CMX-MicroNet CodeWarrior projects; it is the primary working directory for TCP/IP stack development. It contains two separate CodeWarrior projects that are required for developing a CMX-MicroNet TCP/IP stack: *netlibe_ne64.mcp* and *examplee_ne64.mcp*.

More information on *examplee_ne64.mcp* and *netlibe_ne64.mcp* and their individual files is provided in the following sections. Figure 8 shows both of these projects opened in the CodeWarrior IDE (integrated development environment).



File	Code	Data	
Startup Code	80	24	•
Start12.c	80	24	•
Libs	36K	5K	•
netlibe_ne64.lib	17427	4104	
Datapage.c	148	0	•
ansib.lib	20224	2012	
Prm Files	0	0	
_MC9S12NE64_BAN...	n/a	n/a	
Sources	869	63	•
projectvectors.c	125	0	•
example.c	385	35	•
CALLBACK.c	359	28	•
webpage	5K	0	•
PRESS.c	280	0	•
BENCHMRK.c	258	0	•
CMXLOGO.c	2488	0	•
CONTACT.c	252	0	•
HOME.c	218	0	•
index.c	1140	0	•
PAGE2.c	724	0	•
ABOUT.c	266	0	•
16 files	43K	6K	

File	Code	Data	
sources	14K	2K	•
ethernet.c	820	0	•
bootp.c	0	0	•
dhcp.c	0	0	•
Arp.c	1151	44	•
vfile.c	829	277	•
Http.c	2932	730	•
IP.c	1291	22	•
mn_csum.c	348	0	•
MN_PORT.c	168	1607	•
mn_timer.C	124	0	•
mnserver.c	360	0	•
Mnstring.c	413	0	•
ne64_eth.c	754	0	•
Physical.c	77	0	•
Smtp.c	0	0	•
socket.c	1279	344	•
Support.c	405	16	•
Tcp.c	3192	0	•
tftp.c	0	0	•
UDP.c	636	0	•
FTPSERVR.c	0	0	•
Project_Files	2K	3K	•
per_NE64_L19S.c	0	291	•
NE64_API.c	2545	3072	•
debug.c	0	0	•
mnconfig.h	0	0	
projectglobals.h	0	0	
26 files	16K	6K	

Figure 8. CodeWarrior IDE with CMX-MicroNet Projects Open

CMX-MicroNet CodeWarrior Projects

The *netlibe_ne64.mcp* project creates a library (object) file of the CMX-MicroNet API, and the *examplee_ne64.mcp* project contains the specific user application. This section covers these projects and specific files in each project. The specific files that are discussed are the files that will likely require modification for the simple web server described by this application note.

Examplee Project *(examplee_ne64.mcp)* — Contains the source code of the specific user application. The source code in this project includes main() for the TCP/IP stack solution.

- ***example.c*** — Contains the user application including main().
- ***callback.c*** — Among other things, contains settings for:
  - MAC hardware and IP addresses
  - SMTP (simple mail transfer protocol) IP address
  - Gateway IP address
  - Subnet mask
- ***_MC9S12NE64_BANKED.prm*** — Contains Ethernet controller buffer map configuration settings, BUFMAP. The user must modify this file to the user's desired buffer size settings for the Ethernet controller buffers. This setting should match the BUFMAP value in *emac_fifo_cfg.h.*
- Files in webpage directory — Contain HTML, GIF, JPG, Java applets, and other files used in the web server. This directory also contains the files that result when *html2C.exe* is executed (see Table 3).

Netlibe Project *(netlibe_ne64.mcp)* — Creates a library file, *netlibe_ne64.lib*, that *examplee_ne64.mcp* uses. The *netlibe_ne64.mcp* project contains the source code for the CMX-MicroNet API. Most of these files will not require modification during software development. Typically, this project must be modified initially to include the protocols of the CMX-MicroNet project that will be used in the project and to set the initialization settings for the MC9S12NE64 Ethernet controller. After these parameters are set, this project can be compiled to create the *netlibe_ne64.lib* object file.

- ***mn_port.c*** — Contains initialization code for HCS12 modules and other MCU support code. Some of the functions included in *mn_port.c* are:
  - SCI initialization
  - PLL initialization
  - RTI interrupt service routine
  - SCI interrupt service routine
- ***mnconfig.h*** — Contains an interface to set up (turn off or on) and configure various TCP/IP stack protocols (TCP, UDP, ARP, PING, DHCP, etc.) that are used when building the stack.
- ***emac_fifo_cfg.h*** — Contains Ethernet controller buffer map configuration settings, BUFMAP. The user must modify this file to the desired buffer size settings for the Ethernet controller buffers. This setting should match the BUFMAP value in *_MC9S12NE64_BANKED.prm*.
- ***ether_init.h*** — Contains the configuration setup for the MC9S12NE64 integrated MAC and PHY. The user can enable and disable auto-negotiation in this file and also set the auto-negotiation advertisements by modifying this file. Other configuration options are also available.

CMX-MicroNet TCP/IP Stack API

Table 4 provides a brief description of the CMX-MicroNet API functions that are required to develop the simple web server described in this application note. For complete documentation of the CMX-MicroNet API, please reference the CMX-MicroNet user guide.

Table 4. Selected CMX-MicroNet API Functions

Function	Description
mn_init()	Generally sets up the CMX-MicroNet TCP/IP stack. In this example, it calls *mn_arp_init()*, *mn_http_init()*, and *mn_ether_init()*. This routine also sets up the CMX-MicroNet virtual file system. *mn_init()* can be found in *socket.c* in the *netlibe_ne64.mcp* project.
mn_ether_init()	Executes the ETHER_INIT macro in *ethernet.h*. The ETHER_INIT macro defined for this example is *ne64_init()*. *mn_ether_init()* can be found in *ethernet.h* in the *netlibe_ne64.mcp* project.
ne64_init()	Located in *ne64_ether.c* in the *netlibe_ne64.mcp* project. This routine initializes and enables the MC9S12NE64 integrated Ethernet controller.
mn_vf_set_entry(arguments) ***mn_pf_set_entry(arguments)*** ***mn_gf_set_entry(arguments)***	These are virtual file system functions that are located in *vfile.c* in the *netlibe_ne64.mcp* project. These functions are designed to make it easy to retrieve arrays associated with web pages and function pointers used by server-side-includes and HTTP post routines. For *mn_gf_set_entry*, the #define SERVER_SIDE_INCLUDES code in the *mnconfig.h* file must be set to 1.
mn_http_find_value(arguments) ***mn_http_set_file(arguments)*** ***mn_http_set_message(arguments)***	Located in *http.c*. For HTTP functionality, the main web page must be called *index.htm* or *index.html* (unless the main page name is changed in *http.c* by modifying the default_page1 or default_page2 variable names).
mn_ustoa(arguments)	Used to convert an unsigned short integer variable to ASCII. This CMX-MicroNet support function is located in *support.c*.
mn_server()	Located in *mnserver.c*. This routine is a general-purpose server function that combines the HTTP and FTP servers and provides TCP and UDP server functionality. New HTTP and FTP sockets are opened and closed as needed by CMX-MicroNet software. All other sockets must be opened before calling *mn_server*. For example, passive TCP sockets can be opened with the NO_OPEN type before calling *mn_server()*. This function receives a packet and, if an HTTP or FTP packet is received, calls the appropriate HTTP or FTP functions. If the received packet is any type other than HTTP or FTP, the function calls *mn_app_server_process_packet*.

Preparing for CMX-MicroNet TCP/IP Stack Development

This section describes how to prepare for web server development with CMX-MicroNet 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 CMX-MicroNet TCP/IP stack software can be modified and compiled with the CodeWarrior environment. Below are specific details about the development environment and tools required 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.
- 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) — CMX-MicroNet 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.

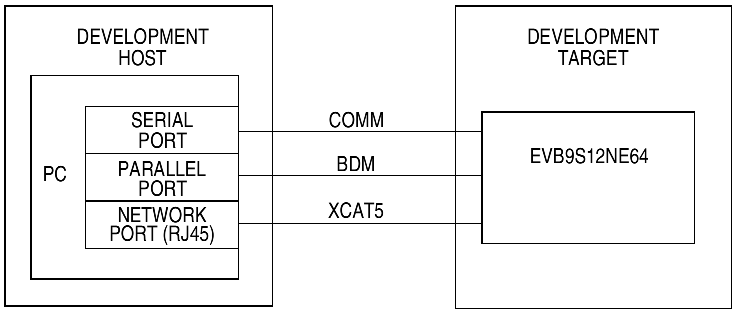


Figure 9. Connecting the Evaluation Board to a PC

This development setup should be used until the CMX-MicroNet application is completed. This 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 *callback.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 CMX-MicroNet 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 192.168.1.1 to 192.168.1.2
- CMX-MicroNet code is programmed with an IP address of:
  - 192.168.1.1, for the development host (PC)
  - 192.168.1.2, for the development target (embedded device)

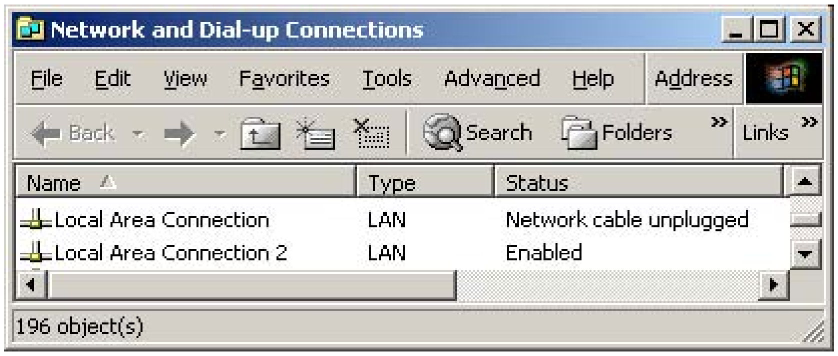
NOTE

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 CMX-MicroNet code, IP addresses and MAC hardware addresses are configured in *callback.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 CMX-MicroNet 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.



Name	Type	Status
Local Area Connection	LAN	Network cable unplugged
Local Area Connection 2	LAN	Enabled

196 object(s)

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 192.168.1.1.

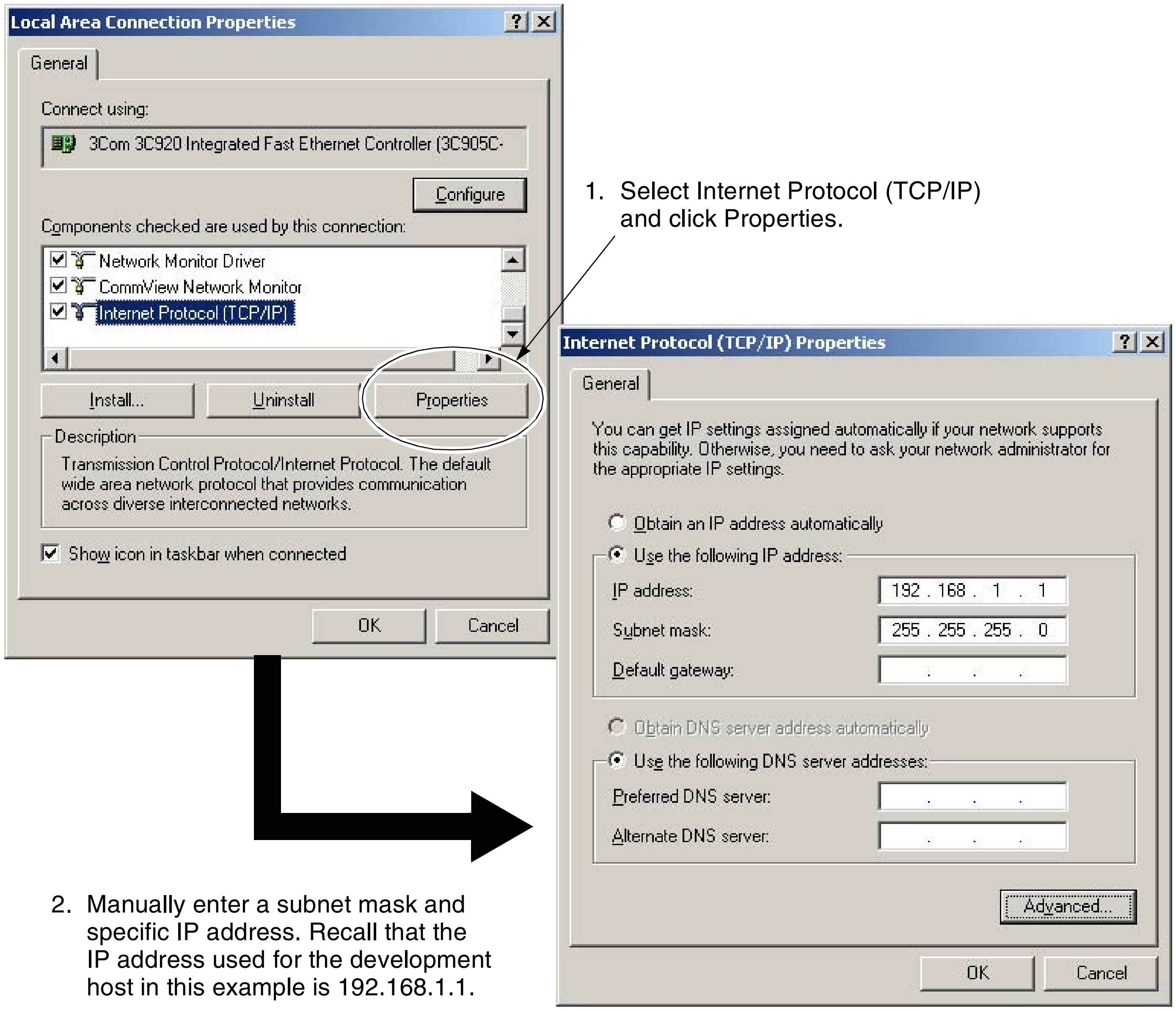

Local Area Connection Properties
General
Connect using:
3Com 3C920 Integrated Fast Ethernet Controller (3C905C-
Configure
Components checked are used by this connection:
Network Monitor Driver
CommView Network Monitor
Internet Protocol (TCP/IP)
Install...
Uninstall
Properties
Description
Transmission Control Protocol/Internet Protocol. The default wide area network protocol that provides communication across diverse interconnected networks.
Show icon in taskbar when connected
OK
Cancel
1. Select Internet Protocol (TCP/IP) and click Properties.
Internet Protocol (TCP/IP) Properties
General
You can get IP settings assigned automatically if your network supports this capability. Otherwise, you need to ask your network administrator for the appropriate IP settings.
Obtain an IP address automatically
Use the following IP address:
IP address: 192 . 168 . 1 . 1
Subnet mask: 255 . 255 . 255 . 0
Default gateway:
Obtain DNS server address automatically
Use the following DNS server addresses:
Preferred DNS server:
Alternate DNS server:
Advanced...
OK
Cancel
2. Manually enter a subnet mask and specific IP address. Recall that the IP address used for the development host in this example is 192.168.1.1.

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."

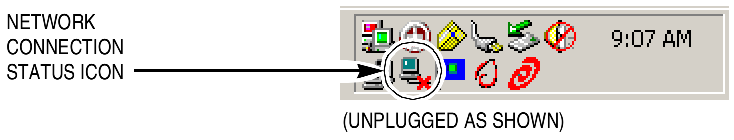


Figure 12. Task Bar Showing Link Status

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 CMX-MicroNet 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

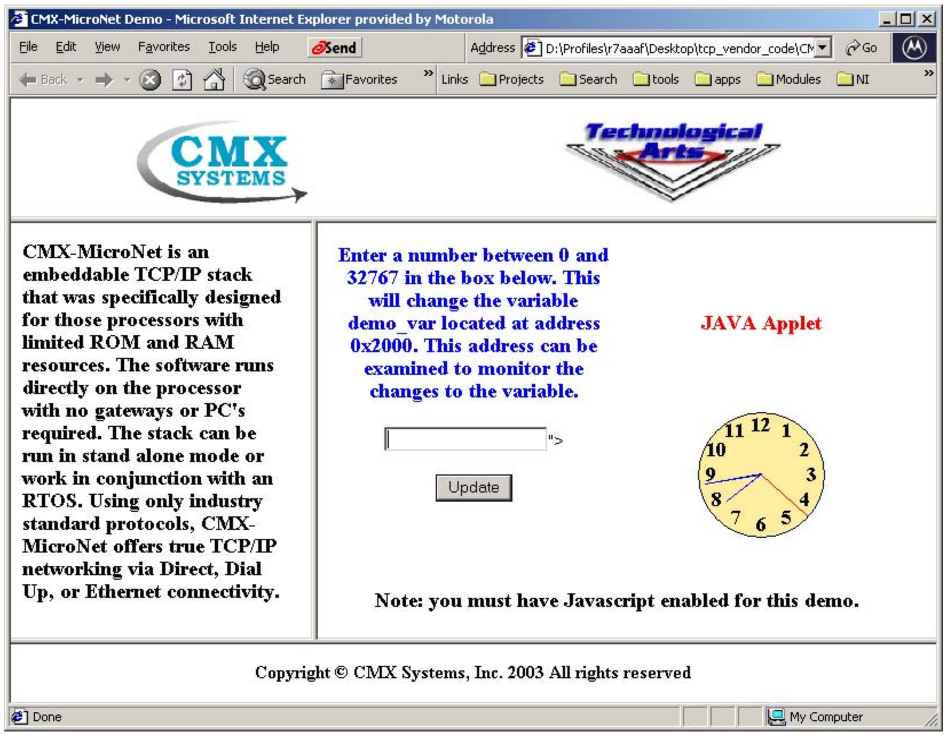
A network protocol analyzer is a powerful and useful tool for network debugging. The network protocol analyzer enables more visibility of packet traffic on the network connection. A network protocol analyzer is used to monitor the connectivity of the Internet or a local area network (LAN).

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 CMX-MicroNet TCP/IP Stack

This section provides an overview of a web server developed with CMX-MicroNet software. This simple web server was developed for distribution with the EVB9S12NE64 as an S-record file. Figure 13 shows the web server.


CMX-MicroNet Demo - Microsoft Internet Explorer provided by Motorola
File Edit View Favorites Tools Help Send
Address D:\Profiles\r7aaaf\Desktop\tcp_vendor_code\CM
Go
Back Search Favorites Links Projects Search tools apps Modules NI
CMX SYSTEMS
Technological Arts
JAVA Applet
Update
Done
My Computer

CMX-MicroNet is an embeddable TCP/IP stack that was specifically designed for those processors with limited ROM and RAM resources. The software runs directly on the processor with no gateways or PC's required. The stack can be run in stand alone mode or work in conjunction with an RTOS. Using only industry standard protocols, CMX-MicroNet offers true TCP/IP networking via Direct, Dial Up, or Ethernet connectivity.

Enter a number between 0 and 32767 in the box below. This will change the variable demo_var located at address 0x2000. This address can be examined to monitor the changes to the variable.

">

Note: you must have Javascript enabled for this demo.

Copyright © CMX Systems, Inc. 2003 All rights reserved

Figure 13. CMX-MicroNet Web Server

This section will overview the files that were modified during development of the CMX-MicroNet web server demo. Acronyms and terms used in this section are defined in Table 1.

Web Pages

The HTML code for the main frame of the web page shown in Figure 13 is provided in this section. Web page development can be assisted with a tool, such as FrontPage by Microsoft, but it is not required. The web page can contain standard HTML components including, but not limited to:

- Frames
- Tables
- Forms
- Embedded Java script
- Applets

When these web components are completed and tested, the next step is to convert them to the equivalent CMX-MicroNet C files using CMX's html2C utility.

Web Server index.htm HTML Source Code:

```
<html><head>
<SCRIPT LANGUAGE="JavaScript">
function checkInfo(form) {
var ok = true;
var valid;
var temp;
if (form.webvar.value == "")
        ok = false;
if (ok)
{
        valid = "0123456789";
        for (var i=0; i<5; i++)
        {
        temp = "" + form.webvar.value.substring(i, i+1);
        if (valid.indexOf(temp) == "-1")
         ok = false;
}
}
if (ok)
{
        var check_num = parseInt(form.webvar.value,10);
        ok = (!isNaN(check_num) && (check_num >= 0) && (check_num <= 32767));
}
if (!ok)
{
        alert("Invalid number entered.\r\nPlease enter a number between 0 and 32767.");
        form.webvar.focus();
        return false;
}
return (ok);
}
</SCRIPT></head>
<BODY text="#000000" vlink="#990099" alink="#990099" bgcolor="#FFFFFF" link="#0000CC"">
<CENTER><table align="center"><tr>
<td align="center" width="50%"><FONT SIZE="+1" COLOR="BLUE"><B>Enter a number between 0 and
32767 in the box below.
This will change the variable demo_var located at address 0x2000. This address can be
examined to monitor the changes to the variable.</B></FONT></td>
<td align="center" width="50%"><FONT SIZE="+1" COLOR="RED"><B>JAVA Applet</B></FONT></td>
</TR><tr>
<td align="center"><FORM NAME="var_info" ACTION="set_demo_var" METHOD=POST onSubmit="return
checkInfo(var_info)">
<INPUT NAME="webvar" TYPE="Text" MAXLENGTH=5 VALUE="<!--#exec cgi="get_demo_var"-->"><P>
<INPUT TYPE="Submit" VALUE="Update"></FORM></td>
<TD ALIGN="center" width="50%"><applet code="JavaCl.class" width="120" height="120">
</applet></TD></tr></table>
<br><br><FONT SIZE="+1"><B>Note: you must have Javascript enabled for this demo.</B></FONT>
</CENTER></body></html>
```

examplee.c

examplee.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 CMX-MicroNet TCP/IP stack then waits and serves a web page on request.

Note that *mn_init()* must be called before using any other CMX-MicroNet function. Refer to *CMX-MicroNet TCP/IP Stack API* for more information about the CMX-MicroNet API functions used in the provided code.

examplee.c Source Code:

```
/**********************************************************
Copyright (c) CMX Systems, Inc. 2003. All rights reserved
*********************************************************/

#include "micronet.h"

/* put #includes for web pages here */
#include "index.h"
#include "cmxlogo.h"
#include "bot.h"
#include "head.h"
#include "main1.h"
#include "side.h"
#include "ta7rssmall.h"
#include "analogcl.h"
#include "javacl.h"

/* Local functions */
void set_demo_var_func(PSOCKET_INFO socket_ptr) cmx_reentrant;
word16 get_demo_var_func(byte **) cmx_reentrant;

#define MSG_BUFF_SIZE   17
byte msg_buff[MSG_BUFF_SIZE];

/* Global variable to be set by web page. */
#pragma DATA_SEG DEMO_MEM
int demo_var;
#pragma DATA_SEG DEFAULT

int main(void)
{
   /* call mn_init before using any other MicroNet API functions */
   if (mn_init() < 0)
      EXIT(1);

   /* Add web pages to virtual file system.
      The main page MUST be called index.htm or index.html.
   */
   mn_vf_set_entry((byte *)"index.htm", INDEX_SIZE, index_htm,VF_PTYPE_STATIC);
   mn_vf_set_entry((byte *)"head.htm", HEAD_SIZE, head_htm,VF_PTYPE_STATIC);
   mn_vf_set_entry((byte *)"side.htm", SIDE_SIZE, side_htm,VF_PTYPE_STATIC);
   mn_vf_set_entry((byte *)"main1.htm", MAIN1_SIZE, main1_htm,VF_PTYPE_STATIC);
   mn_vf_set_entry((byte *)"bot.htm", BOT_SIZE, bot_htm,VF_PTYPE_STATIC);
   mn_vf_set_entry((byte *)"cmxlogo.gif", CMXLOGO_SIZE, cmxlogo_gif,VF_PTYPE_STATIC);
   mn_vf_set_entry((byte *)"ta7rsSmall.jpg", TA7RSSMALL_SIZE, ta7rssmall_jpg,VF_PTYPE_STATIC);
   mn_vf_set_entry((byte *)"JavaCl.class", JAVACL_SIZE, javacl_class,VF_PTYPE_STATIC);
        mn_vf_set_entry((byte *)"AnalogCl.class", ANALOGCL_SIZE,
analogcl_class,VF_PTYPE_STATIC);

   /* add post functions to be used with forms */
   mn_pf_set_entry((byte *)"set_demo_var", set_demo_var_func);

   /* add any get functions (server-side-includes) here */
   mn_gf_set_entry((byte *)"get_demo_var", get_demo_var_func);

   memset(msg_buff,0,sizeof(msg_buff));
   demo_var = 12345;

   mn_server();        /* see mnserver.c */

   return(0);
}

/* ---------------------------------------------------------------------- */

static byte post_var[] = "webvar";
static byte main_page[] = "main1.htm";

/* this function is called from a web page by an HTTP POST request */
void set_demo_var_func(PSOCKET_INFO socket_ptr)
cmx_reentrant {
   VF_PTR vf_ptr;

   /* msg_buff will have decoded value, if available */
   if (mn_http_find_value(BODYptr,post_var,msg_buff))
      {
      demo_var = atoi(msg_buff);

      /* In this example we are always returning main1.htm. */
      vf_ptr = mn_vf_get_entry(main_page);
      if ((vf_ptr == PTR_NULL) ||!(mn_http_set_file(socket_ptr,vf_ptr)))
         {
         /* page was deleted or in the process of being updated.
            send Not Found message.
         */
         mn_http_set_message(socket_ptr,HTTPStatus404,STATUS_404_LEN);
}
}
}

word16 get_demo_var_func(byte **str)
cmx_reentrant {
   *str = msg_buff;
   return ((word16)mn_ustoa(msg_buff, (word16)demo_var));
}
```

callback.c

An excerpt of the source code for *callback.c* is provided. This section of *callback.c* is important because it includes the MAC hardware and IP address settings. This file must be configured correctly. Review *Configuring the MAC Hardware and IP Addresses* for details. The list below provides a simplified description of the code provided.

- ip_dest_addr[IP_ADDR_LEN] — If HTTP or FTP is not being used, an IP address for a destination node must be provided. If HTTP or FTP is used, this variable can be ignored. HTTP and FTP are configured in *mnconfig.h.*
- ip_src_addr[IP_ADDR_LEN] — Embedded device IP address. If DHCP is configured and used, this variable can be ignored. DHCP is configured in *mnconfig.h*.
- byte eth_src_hw_addr[ETH_ADDR_LEN] — This MAC hardware address for the embedded device should be a unique 48-bit number as specified by IEEE.
- byte eth_dest_hw_addr[ETH_ADDR_LEN] — If ARP is not being used, a MAC hardware address for a destination node must be provided in this variable.

The SMTP server IP address can also be set up in *callback.c* if the CMX-MicroNet SMTP client is used. SMTP is used for sending email on the Internet. For this example, SMTP is not used.

Excerpt from callback.c:

```
#if Ethernet
byte ip_dest_addr[IP_ADDR_LEN] = {192,168,2,2};
#if (PING_GLEANING)
byte ip_src_addr[IP_ADDR_LEN] = {0,0,0,0};
#else
byte ip_src_addr[IP_ADDR_LEN] = {192,168,2,3}; // static IP address of embedded device
#endif      /* PING_GLEANING */
#else
byte ip_dest_addr[IP_ADDR_LEN] = {192,6,94,5};
#if (PING_GLEANING)
byte ip_src_addr[IP_ADDR_LEN] = {0,0,0,0};
#else
byte ip_src_addr[IP_ADDR_LEN] = {192,6,94,2};
#endif      /* PING_GLEANING */
#endif      /* Ethernet */

#if (SMTP)
/* replace the ip address below with the ip address of your SMTP server */
byte ip_smtp_addr[IP_ADDR_LEN] = {216,148,227,71};
#endif      /* (SMTP) */

#if Ethernet
/************************************************************
   if using a chip with EEPROM you may need to write a routine
   to take the value of the hw_addr in EEPROM and put it into
   the array below on startup, otherwise replace eth_src_hw_addr
   below with the proper Ethernet hardware address.
************************************************************/
byte eth_src_hw_addr[ETH_ADDR_LEN] = {0x00,0x00,0x12,0x34,0x56,0x78};

/************************************************************
```

```
   If ARP is used the array below is used as a temporary holder
   for the destination hardware address. It does not have to be
   changed.

   If ARP is not being used replace the hardware address below
   with the hardware address of the destination. The hardware
   address used MUST be the correct one.
**************************************************************/
byte eth_dest_hw_addr[ETH_ADDR_LEN] = {0x00,0xE0,0x98,0x03,0xE5,0xFA};

/*************************************************************
   If a gateway is being used set the gateway IP address and
   subnet mask below.

   If a gateway is not being used:
      set the gateway IP address to {255,255,255,255}
      set the subnet mask to        {255,255,255, 0}
**************************************************************/
byte gateway_ip_addr[IP_ADDR_LEN] = {255,255,255,255};
byte subnet_mask[IP_ADDR_LEN]     = {255,255,255, 0};

#endif      /* Ethernet */
```

mnconfig.h

Editing *mnconfig.h* is required to select the protocols used, number of interfaces, number of sockets, sizes of transmit and receive buffers, etc. To minimize code size, only the protocols that are required for the application should be used. An excerpt of *mnconfig.h* is provided. Important settings for the web server example are:

- Enable TCP (enabling UDP is optional)
- Enable Ethernet
- Enable PING
- Enable ARP
- Disable DHCP
- Enable HTTP and SERVER_SIDE_INCLUDES
- Enable VIRTUAL_FILE

In *mnconfig.h*, defining a value of 1 enables a feature, and a value of 0 disables a feature. When editing *mnconfig.h*, recall that some protocols are dependent on each other. For instance, the HTTP requires the TCP. In addition, for CMX-MicroNet software, either UDP or TCP must be enabled for compilation. Figure 14 shows several popular network protocols and their dependence.

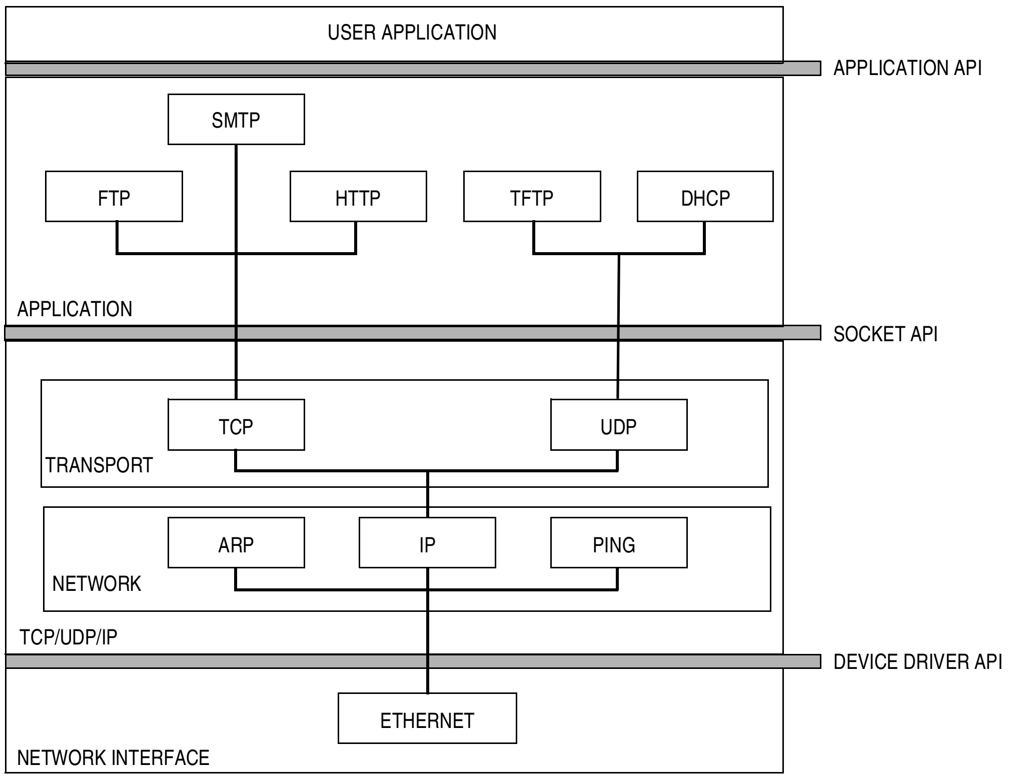


Figure 14. Popular Network Protocols and Their Dependence

Excerpt from mnconfig.h

```
/*********************************************************
Copyright (c) CMX Systems, Inc. 2004. All rights reserved
*********************************************************/

#ifndef MNCONFIG_H_INC
#define MNCONFIG_H_INC 1

/* Protocols */
#define TCP         1
#define UDP         1
#define UDP_CHKSUM  1
#define ETHERNET    1
#define SLIP        0
#define PPP         0
#define PING        1
#define IGMP        0

/* Sockets */
#define NUM_SOCKETS       5
#define SOCKET_WAIT_TICKS 600
#define RECV_BUFF_SIZE    900
#define XMIT_BUFF_SIZE    570
#define SOCKET_INACTIVITY_TIME    0

/* TCP/IP options */
```

```
#define TIME_TO_LIVE     64
#define TCP_WINDOW       512
#define TCP_RESEND_TICKS 600
#define TCP_RESEND_TRYS  12
#define PING_GLEANING    0
#define PING_BUFF_SIZE   32
#define ALLOW_BROADCAST  0
#define ALLOW_MULTICAST  0
#define MULTICAST_TTL    1
#define IGMP_LIST_SIZE   4

/* Ethernet */
#define POLLED_ETHERNET  0
#define ETHER_WAIT_TICKS 5

/* ARP */
#define ARP              1
#define ARP_WAIT_TICKS   600
#define ARP_TIMEOUT      0
#define ARP_AUTO_UPDATE  0
#define ARP_CACHE_SIZE   4
#define ARP_KEEP_TICKS   6000
#define ARP_RESEND_TRYS  6

/* DHCP */
#define DHCP                     0
#define DHCP_RESEND_TRYS         4
#define DHCP_DEFAULT_LEASE_TIME  36000

/* BOOTP */
#define BOOTP              0
#define BOOTP_RESEND_TRYS  6
#define BOOTP_REQUEST_IP   1

/* PPP options */
#define USE_PAP             1
#define PAP_USER_LEN        10
#define PAP_PASSWORD_LEN    10
#define PAP_NUM_USERS       1
#define PPP_RESEND_TICKS    300
#define PPP_RESEND_TRYS     6
#define PPP_TERMINATE_TRYS  2
#define FAST_FCS            1

/* Modem */
#define MODEM           0
#define DIRECT_CONNECT  1
#define NULL_MODEM      1
#define REMOTE_IS_NT    1
#define USE_PASSWORD    0

/* HTTP */
#define HTTP                   1
#define SERVER_SIDE_INCLUDES   1
#define INCLUDE_HEAD           0
#define URI_BUFFER_LEN         52
#define BODY_BUFFER_LEN        52
#define HTTP_BUFFER_LEN        512
```

```
/* FTP */
#define FTP                0
#define FTP_SERVER         1
#define FTP_MAX_PARAM      24
#define FTP_BUFFER_LEN     512
#define FTP_USER_LEN       10
#define FTP_PASSWORD_LEN   10
#define FTP_NUM_USERS      1
#define NEED_MEM_POOL      0
#define MEM_POOL_SIZE      4096

/* TFTP */
#define TFTP               0
#define TFTP_RESEND_TRYS   3
#define TFTP_USE_FLASH     1

/* SMTP */
#define SMTP              0
#define SMTP_BUFFER_LEN   512

/* Virtual File System */
#define VIRTUAL_FILE     1
#define NUM_VF_PAGES     6
#define VF_NAME_LEN      20
#define FUNC_NAME_LEN    20
#define NUM_POST_FUNCS   5
#define NUM_GET_FUNCS    5

#endif   /* ifndef MNCONFIG_H_INC */
//***************************************************************
```

Ether_Init.h

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

- AUTO_NEGOTIATE — Sets up *ne64_init()* to initialize the NE64 in auto-negotiation mode if asserted. If not asserted, the developer must manually set up the speed and duplex for the NE64 using the *full_duplex* and *speed_100* variables.
- HALF10, FULL10, HALF100, and FULL100 — Set the auto-negotiation advertisements and should be set to 1 if that mode is to be advertised in auto-negotiation.
- SPEED100 — if auto-negotiation is disabled (AUTO_NEG = 0), this bit manually sets the device link speed.
- FULL_DUPLEX — if auto-negotiation is disabled (AUTO_NEG = 0), this bit manually sets the device link speed.
- ETYPE_PET, ETYPE_EMW, ETYPE_IPV6, ETYPE_ARP, ETYPE_IPV4, ETYPE_IEEE, and ETYPE_ALL — Set the EMAC Ethertype filtering modes.
- BRODC_REJ, CON_MULTIC, and PROM_MODE — Set the EMAC MAC address filtering modes.
- PTIME — EMAC flow control setting.
- RX_MAX_FL — Sets a maximum frame length by assigning a value to this identifier.

Excerpt from Ether_Init.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
/* what I advertise */
#define HALF100   1           /**< Configure mode that the device should advertise (advertise=1
& not= 0...   */
#define FULL100   1           /**< Configure mode that the device should advertise (advertise=1
& not= 0...   */
#define HALF10    1           /**< Configure mode that the device should advertise (advertise=1
& not= 0...   */
#define FULL10    1           /**< Configure mode that the device should advertise (advertise=1
& not= 0...   */

#else          /* AUTO_NEG */
#define SPEED100     0    /**< 1 - enable 100 MBps / 0 - enable 10 MBps */
#define FULL_DUPLEX 0    /**< 1 - enable full duplex / 0 - disable     */
#endif         /* AUTO_NEG */

/* EMAC CONTROL */
/* ============== */
/* Address Filtering; RXMODE setting: PAUSE frame supported, Accept Unique, Brodcast, MultiCast
*/
#define   BRODC_REJ    0        /**<  1 = All broadcast address frames are rejected.
*/
#define   CON_MULTIC    0         /**<  1 = Multicast hash table is used for checking multicast
addresses. */
#define  PROM_MODE   0     /**< 1 = All frames are received regardless of address.
*/

/* Ethertype Control */
#define ETYPE_PET    0 /**<  1 = accept Programmable Ethertype, 'etype' parameter is used */
#define ETYPE_EMW    0 /**<  1 = accept Emware Ethertype  */
#define ETYPE_IPV6   0 /**<  1 = accept Internet IP version (IPV6) Ethertype */
#define ETYPE_ARP    1 /**<  1 = accept Address Resolution Protocol (ARP) Ethertype */
#define ETYPE_IPV4   1 /**<  1 = accept Internet IP version 4 (IPV6) Ethertype */
#define ETYPE_IEEE   0 /**<  1 = accept IEEE802.3 Length Field Ethertype     */
#define ETYPE_ALL    0 /**<  1 = accept Accept all ethertypes. THIS OVERRIDES OTHER SETTINGS */

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

/* Recieve maximum frame length                                                        */
#define RX_MAX_FL  1536    /**< Receive maximum frame length    */
```

emac_fifo_cfg.h

An excerpt of the source code for *emac_fifo_cfg.h* is provided. The code shows configurations for the EMAC Ethernet buffer allocation.

For the MC9S12NE64, 8K of RAM is available and shared between user RAM and the EMAC Ethernet buffer space. The EMAC Ethernet buffer space consists of three buffers:

- One transmit buffer
- Two receive buffers

Each buffer is designed to hold only one Ethernet frame. Table 5 provides the configurations for the shared RAM usage within the MC9S12NE64 MCU.

Table 5. Shared RAM Usage

BUFMAP	Individual Buffer Size (Bytes)	Total Size of EMAC Ethernet Buffer Space (Bytes)	Remainder RAM for User Application (Bytes)
0	128	384 = 0.375K	7.625K
1	256	768 = 0.75K	7.25K
2	512	1536 = 1.5K	6.5K
3	1K	3072 = 3K	5K
4	1.5K	4608 = 4.5K	3.5K

Because the maximum size of an Ethernet frame is approximately 1.5 Kbytes, a setting of BUFMAP = 4 would allow each of the three MC9S12NE64 buffers to hold one frame of the maximum allowable size as dictated by the IEEE 802.3 specification. Settings for BUFMAP less than 4 are provided to:

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

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

Setting BUFMAP to create a buffer size limit reduces the burden of the CPU by ignoring packets that are too large. If a receive frame exceeds the receive buffer size, the frame is not accepted, and neither the receive complete flag nor the receive error flag is set. No CPU bandwidth is used because the EMAC state machine does all packet filtering.

Excerpt from emac_fifo_cfg.h:

```
/******************************************************************************
 *
 * Freescale Semiconductor, Inc. 2003 All rights reserved
 *
 ******************************************************************************
 *
 * $File Name     : emac_fifo_cfg.h$
 * Description    : definition of EMAC Ethernet buffer allocation
 *
 *
 * $Version       : 1.0.3.0$
 * $Date          : Sep-23-2003$
 *
 * $Last Modified By : r29303$
 *
 ******************************************************************************/

#ifndef EMAC_FIFO_CFG_H
#define EMAC_FIFO_CFG_H

#define BUFMAP 4

#if BUFMAP > 4
#error Illegal FIFO buffer size
#endif

#if BUFMAP == 0
#define EMAC_RX_SZ 128
#define EMAC_TX_SZ 128

#elif BUFMAP == 1
#define EMAC_RX_SZ 256
#define EMAC_TX_SZ 256

#elif BUFMAP == 2
#define EMAC_RX_SZ 512
#define EMAC_TX_SZ 512

#elif BUFMAP == 3
#define EMAC_RX_SZ 1024
#define EMAC_TX_SZ 1024

#elif BUFMAP == 4
#define EMAC_RX_SZ 1536
#define EMAC_TX_SZ 1536

#endif

#endif      /* EMAC_FIFO_CFG_H */
```

_MC9S12NE64_BANKED.prm.h

An excerpt of the source code for *_MC9S12NE64_BANKED.prm* is provided. The code shows configurations for the EMAC Ethernet buffer allocation. The RAM segments definition should match the corresponding configurations of BUFMAP in *emac_fifo_cfg.h*.

Excerpt from _MC9S12NE64_BANKED.prm:

```
SEGMENTS

//     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) */
//     RAM        = READ_WRITE 0x3200 TO 0x3FFE;      /* BUFMAP = 4 (1.5K) */
```

CMX-MicroNet Project Configuration to Optimize the Stack Solution

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

- Minimize web page content
- Modify *mnconfig.c* to use only required network protocols
- Set buffers to appropriate values

Minimize 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 page features. Each web page graphic, for example, can easily require 6 to 8 Kbytes of FLASH.

Modify *mnconfig.h* to Use Only Required Network Protocols

See the mnconfig.h section for details. If using Ethernet without DHCP or BOOTP, these protocols should not be enabled in the stack. A complete TCP/IP stack consists of a large set of networking protocols that require large memory and CPU resources. 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 15 illustrates a simplified or partial TCP/IP stack implementation. This stack uses only the UDP protocol and the user applications.

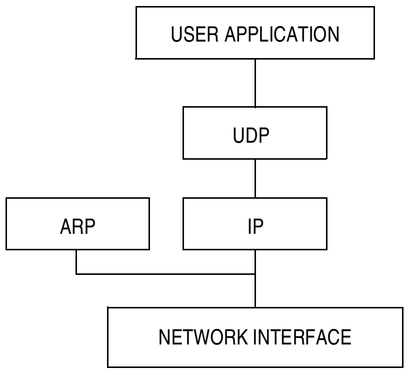


Figure 15. Partial Stack TCP/IP Stack Implementation

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.

CMX-Micronet Zero Copy Functionality

Zero Copy refers to the TCP/IP stack's ability to send and receive from the Ethernet buffers' stack without copying data to user RAM. CMX's zero copy functionality was introduced in CMX-Micronet version 3.03 to optimize RAM usage with the MC9S12NE64. The zero copy mode of the CMX stack can be disabled if the user chooses. Because the MC9S12NE64 has an integrated Ethernet buffer, there is no need to make a second copy of the Ethernet buffers. Instead, these Ethernet buffers can be directly accessed.

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 user RAM and Ethernet buffer RAM. It is possible that enabling zero copy could cause the HTTP server to drop packets because, in zero copy mode, only two incoming packets will be stored at the same time.

To enable zero copy mode in the CMX-MicroNet TCP/IP stack, the USE_SEND_BUFF and USE_RECV_BUFF variables in *mn_env.h* of the netlib directory must be cleared to 0. If set to 1, CMX-MicroNet makes a copy of data in the Tx and Rx Ethernet buffers in user RAM.

Set Buffer 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 buffer sizes are controlled in *emac_fifo_cfg.h* and *_MC9S12NE64_BANKED.prm*.

Conclusion

Combining the MC9S12NE64 and the CMX-MicroNet stack software provides a single-chip Ethernet system solution that is low cost and easy to use. The CMX-MicroNet 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 Semiconductor document contains information that conflicts with the information in the device user guide, the user guide 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

AN2700
Rev. 0.3, 9/2004

Launched by Motorola
freescale™
semiconductor