Application Note

AN2624/D
1/2004

Basic Web Server Development with the 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 and relatively low price of implementation. Ethernet is ease to use, widely available, and has a scalable and established infrastructure. Ethernet is described by IEEE Standard 802.3™.

With Ethernet, 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.

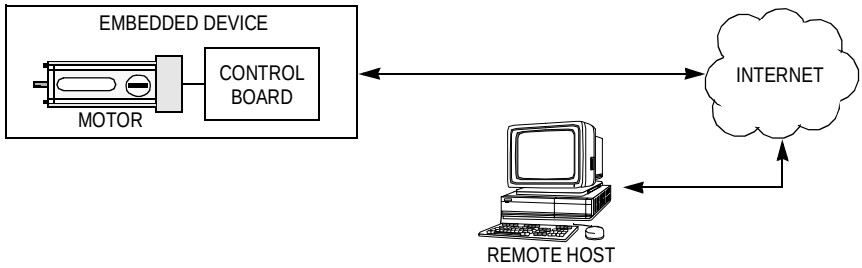


Figure 1. Embedded Device on Internet

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
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.

Connectivity Example Applications

Connectivity systems that obey 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

Scope of the Application Note

This application note details the creation of a basic web server for an embedded device. The components of this embedded web server system include:

- MC9S12E128 microcontroller unit (MCU)
- Technological Arts™ 9S12ADAPT™ MC9S12E128 evaluation board
- Technological Arts LAN interface card
- Motorola stand-alone low-level LAN91C111 Ethernet drivers
- CMX-MicroNet™ TCP/IP stack

The goal of this application note is to help familiarize first-time users of the CMX-MicroNet TCP/IP™ stack with its API and its Metrowerks® CodeWarrior® project organization. To do this, a walk-through of developing a simple web server is provided. This understanding will help speed initial CMX-MicroNet TCP/IP stack software application development. 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.

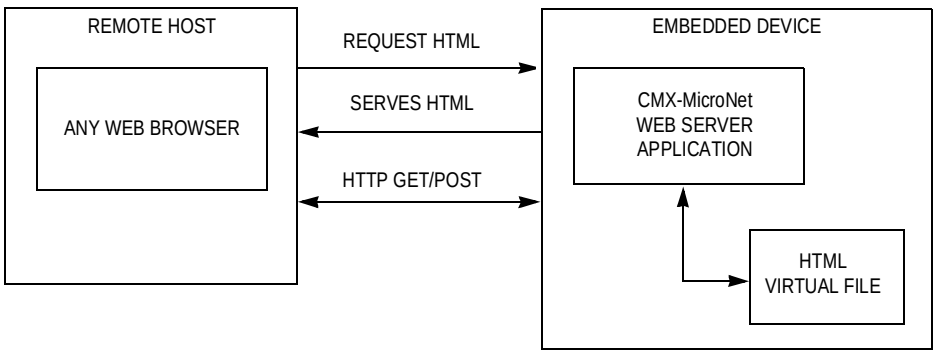


Figure 2. Web Server Example TCP/IP Stack Application

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 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.

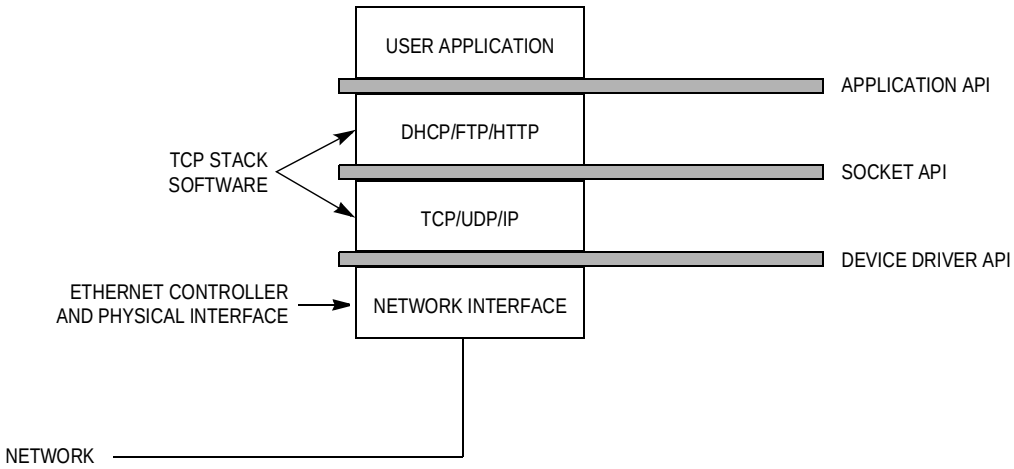


Figure 3. Block Diagram of TCP/IP Model

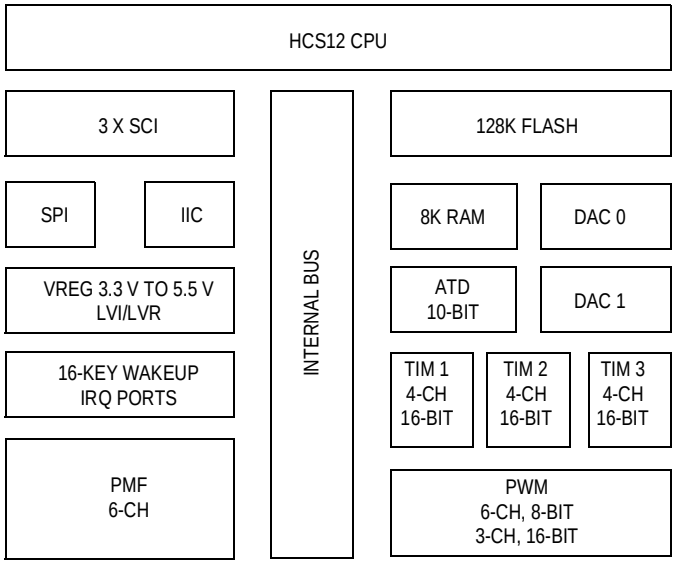
MC9S12E128 and Technological Arts Ethernet Reference Design

This section will briefly describe the MC9S12E128 MCU and the Technological Arts MC9S12E128 Ethernet reference design.

MC9S12E128

The MC9S12E128 is a 16-bit MCU with 8K of RAM and 128K FLASH. It is designed for the low-cost general-purpose distribution market. It is the first of a series of low-cost general-purpose products to address the UPS (uninterruptible power supply) and home appliance markets.

The MC9S12E128 has a rich set of features. In the 112-pin package, an expanded bus is available that can be used to interface external peripherals such as an Ethernet controller. More information on the MC9S12E128 is available from the Motorola website: http://motorola.com/semiconductors. A block diagram of the MC9S12E128 is provided in **Figure 4**.



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 interrupt
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 MC9S12E128

Technological Arts MC9S12E128 Ethernet Reference Design

Technological Arts provides a modular approach to an MC9S12E128 Ethernet reference design, resulting in two stand-alone boards that are connected via a 50-pin header.

Adapt9S12E128™

Adapt9S12E128, shown in **Figure 5**, is a compact and modular MC9S12E128 microcontroller evaluation board that is compatible with other Technological Arts products including several application cards (such as the LAN interface card), prototyping cards, backplanes, and solderless breadboards. The Adapt9S12E128 includes an 8-MHz crystal, reset button, BDM connector, and RS-232C interface.

See the Technological Arts website, http://www.technologicalarts.com, for optional features and more information.

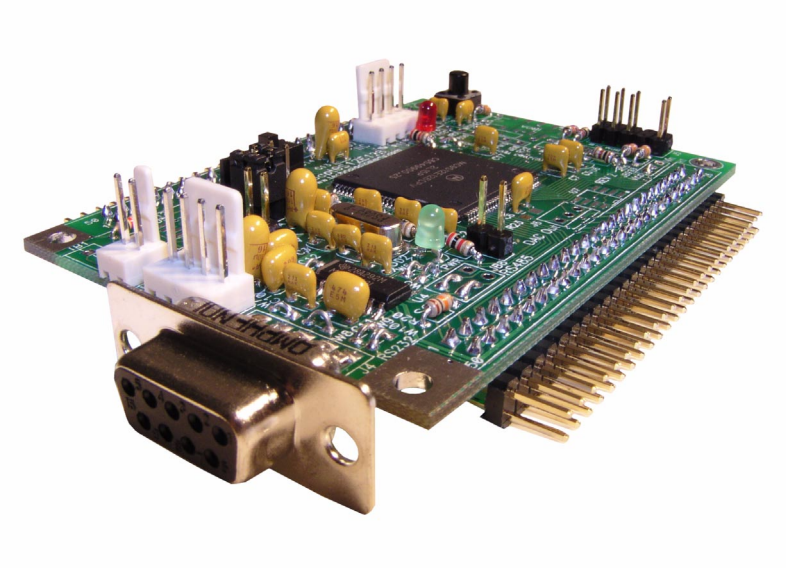


Figure 5. Adapt9S12E128 Evaluation Board from Technological Arts

100-BaseT LAN Interface Card for Adapt9S12

The LAN interface card for Adapt9S12 uses the SMSC LAN91C111 Ethernet controller and works with Adapt9S12E128 in expanded wide mode via the Adapt9S12E128 H2 connector.

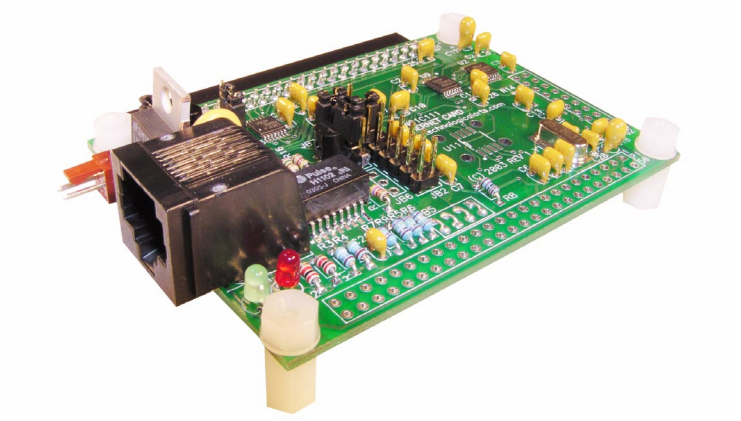


Figure 6. Technological Arts 100-BaseT LAN Interface Card for Adapt9S12

Combining the Two Stand-Alone Boards

Adapt9S12E128 MC9S12E128 evaluation and 100-BaseT LAN interface card for Adapt9S12 boards are interfaced via the 50-pin header, H2. The connected boards are shown in **Figure 7** with the Technological Arts multi-function demo card for Adapt9S12, which plugs into H1. The demo card for Adapt9S12 provides light-emitting diodes (LEDs), speaker, dual in-line (DIP) switches, push-buttons, photocell, thermistor, and dual-logic metal oxide semiconductor field-effect transistors (MOSFETs). The demo card for Adapt9S12 will not be addressed any further in this document.

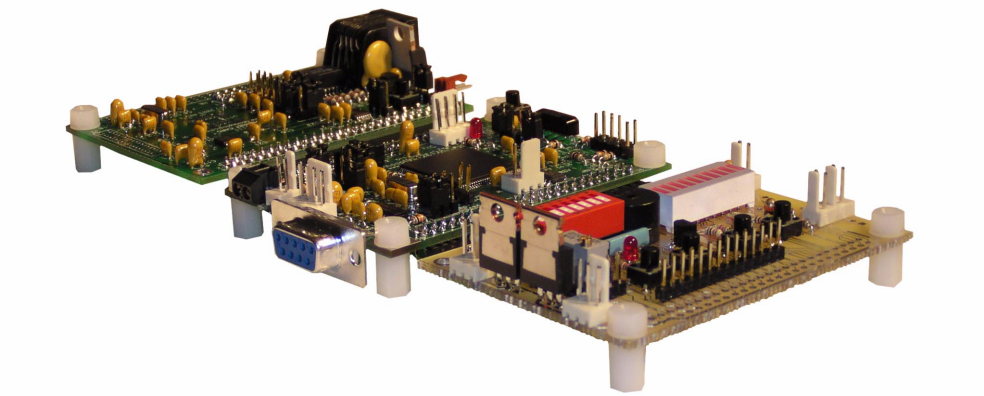


Figure 7. Technological Arts MC9S12E128 Ethernet Reference Design

Board Versions and Settings

The version numbers of the boards used in this application note and their jumper settings are provided in this section.

Table 2. Board Versions

Board	Revision
Adapt9S12E128 MC9S12E128 evaluation board	2.0
100-BaseT LAN Interface Card for Adapt9S12	1.0

Important jumper settings for each board are provided in **Table 3** and **Table 4**. The simplified layouts of the Adapt9S12E128 and 100-BaseT LAN interface card are provided in **Figure 8** and **Figure 9**.

Settings for the Adapt9S12E128

When connecting the boards together, only one of the boards must be powered. In this case, apply power to the Adapt9S12E128.

The Adapt9S12E128 must be configured for normal single-chip mode (MODA=MODB=0, MODC=1) for the CMX-MicroNet stack. In the program, before the Ethernet controller is initialized, the program is set to operate in expanded mode.

For detailed information on this evaluation board, view the Adapt9S12E128 user's manual from the Technological Arts web site.

Table 3. Settings for the Adapt9S12E128

Adapt9S12E128 Jumper/Switch	Jumper Settings
Ch1	Don't Care
JB1, MODB — mode select pin B	2–4
JB1, MODA — mode select pin A	1–2
JB4, $\overline{\text{XIRQ}}$	Off
JB5, U3 pull down	Off
JB6, IR LED shutdown	Don't Care
JB7, IR sensitivity control	Don't Care
JB8	Don't Care
JB9	2–3
JB9 header	V_{CC}
RS-232 select	Don't Care
RS-485	Don't Care
SW2 — switch used for the serial monitor	Run
TERM header	On
VRH select	2–3

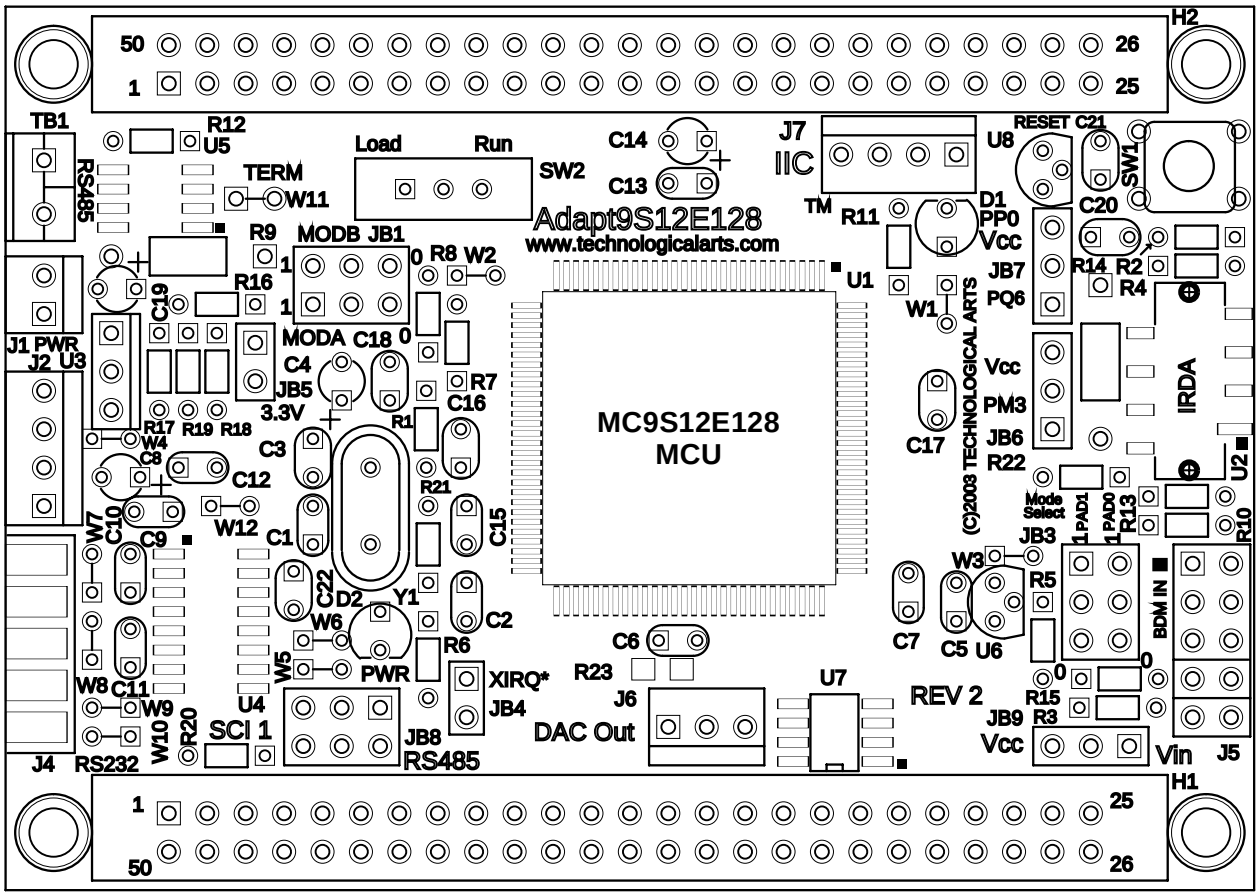


Figure 8. Simplified Layout of the Adapt9S12E128

Settings for the LAN Interface Card

Table 4. Settings for the LAN Interface Card

LAN Interface Card Jumper/Switch	Jumper Settings
JB1 — Latch decoding (XCS/ECLK)	XCS, 1–2
JB2 — LAN decoding	1–2
JB3 — Reset control selection header	1–2
JB4 — Power source selection header	V_{CC}, 2–3
JB7 — Latch decoding	2–3

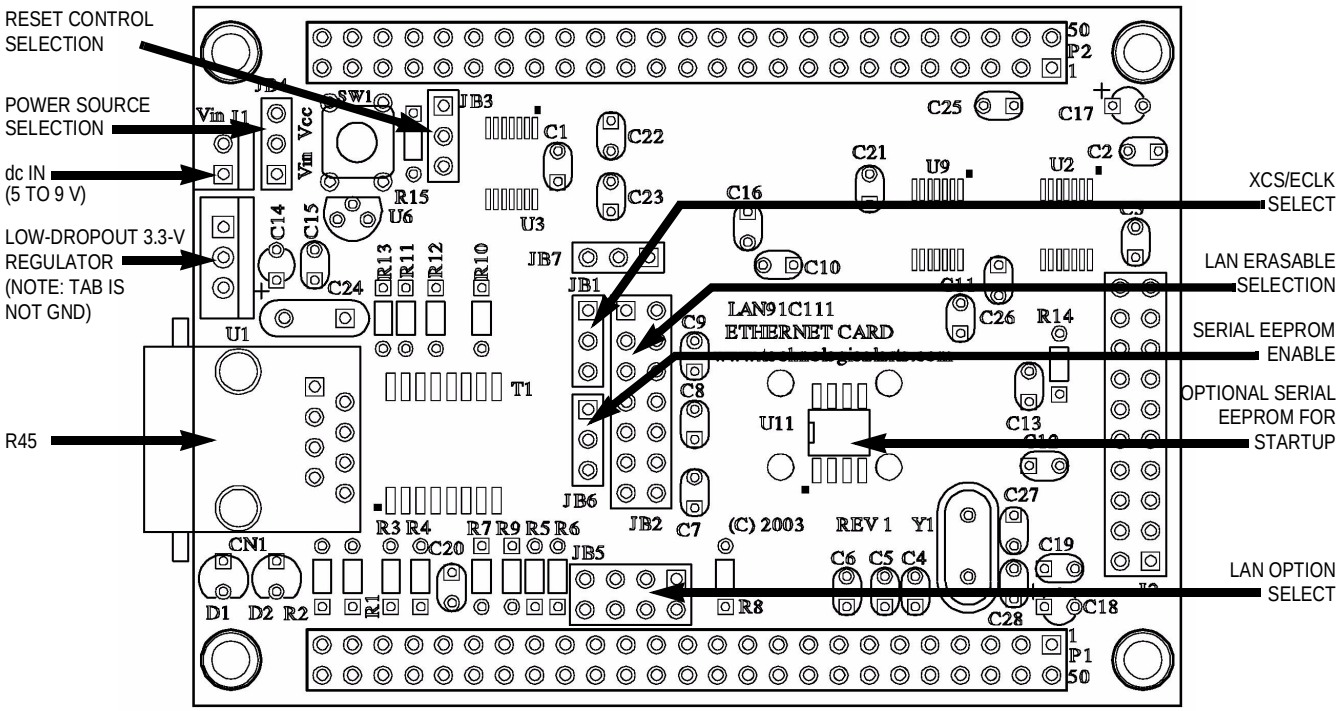


Figure 9. Simplified Layout of 100-BaseT LAN Interface Card

CMX Stack Overview

This sections includes:

- CMX introduction
- Motorola 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
- Ignores all TCP options
- No support to handle fragmented packets

Motorola Low-Level LAN91C111 Ethernet Drivers

Powering the CMX-MicroNet TCP/IP stack is a low-level Ethernet driver for the LAN91C111 Ethernet controller. These drivers are integrated within the CMX-MicroNet TCP/IP stack source code. However, a Motorola stand-alone (without a TCP/IP stack) version of the low-level Ethernet drivers is available.

CMX-MicroNet Installation and Project Organization

CMX delivers CMX-MicroNet source code as an installation file. When this file is installed, several directories are placed on the target PC. **Figure 10** illustrates the CMX-MicroNet project directories that are installed. The CMX-MicroNet main project directory is {Project Directory}\MICRONET, where {Project Directory} is the installation directory.

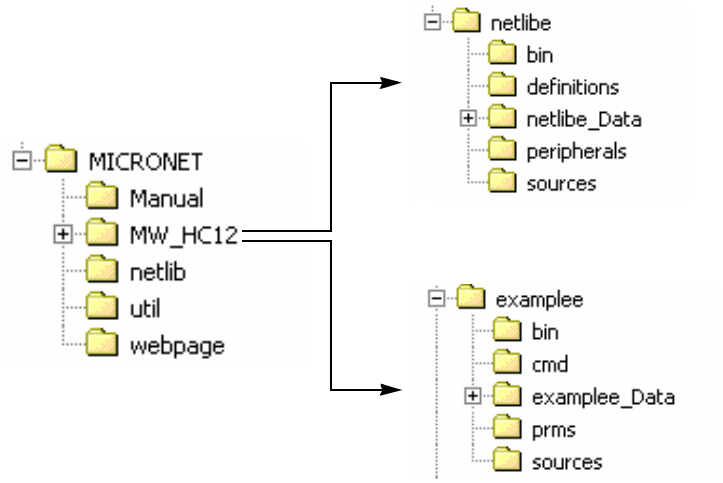


Figure 10. CMX-MicroNet Project Directory Structure

Table 5 is a description of each sub-directory in the CMX-MicroNet main project directory.

Table 5. CMX-MicroNet Project Sub-Directory Descriptions

Sub-Directory	Description
{Project Directory}\MICRONET\Manual	Contains CMX documentation and user manual
{Project Directory}\MICRONET\netlib	Contains the source files for the CMX-MicroNet API
{Project Directory}\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}\MICRONET\Webpage	Contains the HTML, JPG, GIF, and other files developed for the embedded web server
{Project Directory}\MICRONET\Mw_hc12	Contains CMX-MicroNet CodeWarrior projects; it is the primary working directory for TCP/IP stack project development. It contains two separate CodeWarrior projects that are required for developing a CMX-MicroNet-based TCP/IP stack solution: *netlibe.mcp* and *examplee.mcp*.

More information on *examplee.mcp* and *netlibe.mcp* and their individual files is provided in the following sections. **Figure 11** shows both of these projects opened in the CodeWarrior IDE (integrated development environment).

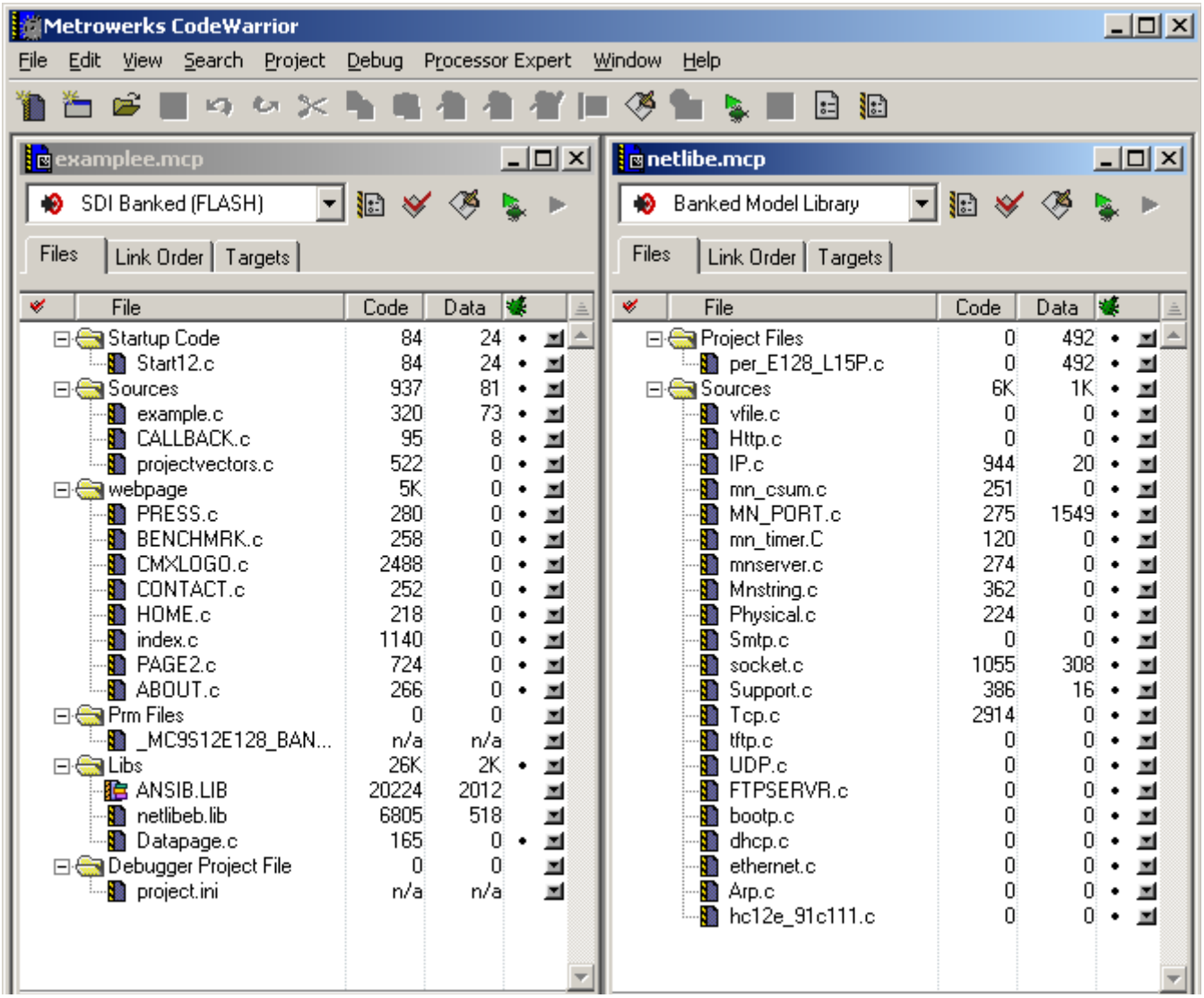


Figure 11. CodeWarrior IDE with CMX-MicroNet Projects Open

CMX-MicroNet CodeWarrior Projects

As previously noted, the *netlibe.mcp* project creates a library (object) file of the CMX-MicroNet API, and the *examplee.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.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
- 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 5**).

Netlibe Project (*netlibe.mcp*) — Creates a library file, *netlibe.lib*, that *examplee.mcp* uses. The *netlibe.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 external Ethernet controller. After these parameters are set, this project can be compiled to create the *netlibe.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.
- ***hcs12e_91C111.c*** — Contains an API that the CMX-MicroNet TCP/IP stack uses to interface to the external Ethernet controller (in this case, the LAN91C111). This file should be modified if you want to change the start-up code for the LAN91C111. For example, the LAN91C111 is configured for auto-negotiation. If this is not desired, *hcs12e_91C111.c* should be modified.

CMX-MicroNet TCP/IP Stack API

Table 6 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 6. 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.mcp* project.
mn_ether_init()	Executes the ETHER_INIT macro in *Ethernet.h*. The ETHER_INIT macro defined for this example is *smsc91C111_init()*. *Mn_ether_init()* can be found in *Ethernet.c* in the *netlibe.mcp* project.
smsc91C111_init()	Located in *hcs12e_91C111.c* in the *netlibe.mcp* project. This routine initializes and enables the external Ethernet controller, SMSC LAN91C111.
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.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*. It 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. 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 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®

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 MC9S12E128 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 12 shows the basic connection of a PC running CodeWarrior software to a Technological Arts MC9S12E128 Ethernet reference board. 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 MC9S12E128 MCU. On the Adapt9S12E128 evaluation board, the BDM connector is labeled BDM IN.
- 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 MC9S12E128 SCI ports. By using Hyperterminal and a serial cable, these debug messages can be captured.

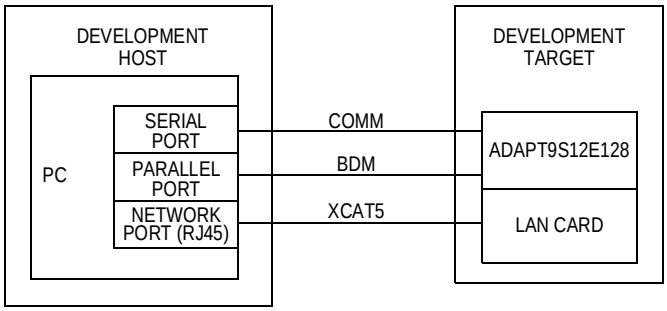


Figure 12. Connecting the Evaluation Board to a PC for Development with CMX-MicroNet

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 to a real network, changes should be required to only the MAC hardware and IP addresses.

Configuring the MAC Hardware and IP Addresses

When the web server is complete, it can then 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. Every network device should 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 Technological Arts MC9S12E128 Ethernet reference board should be assigned by the developer. This address is used by the datalink layer which is implemented by the LAC91C111 Ethernet controller and the low-level drivers. Again, this can be changed 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 developer's 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 the CMX-MicroNet 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 13**.

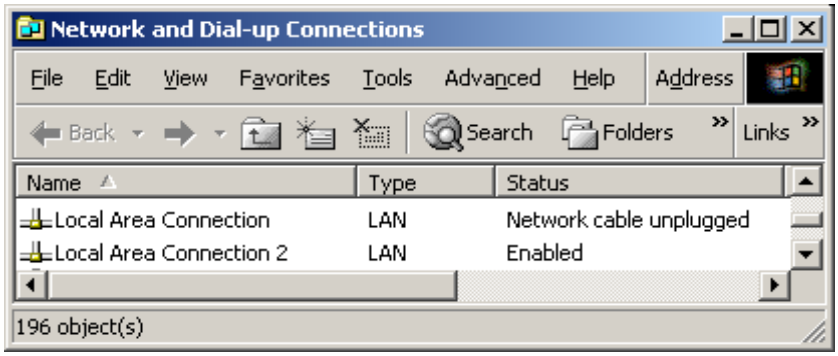


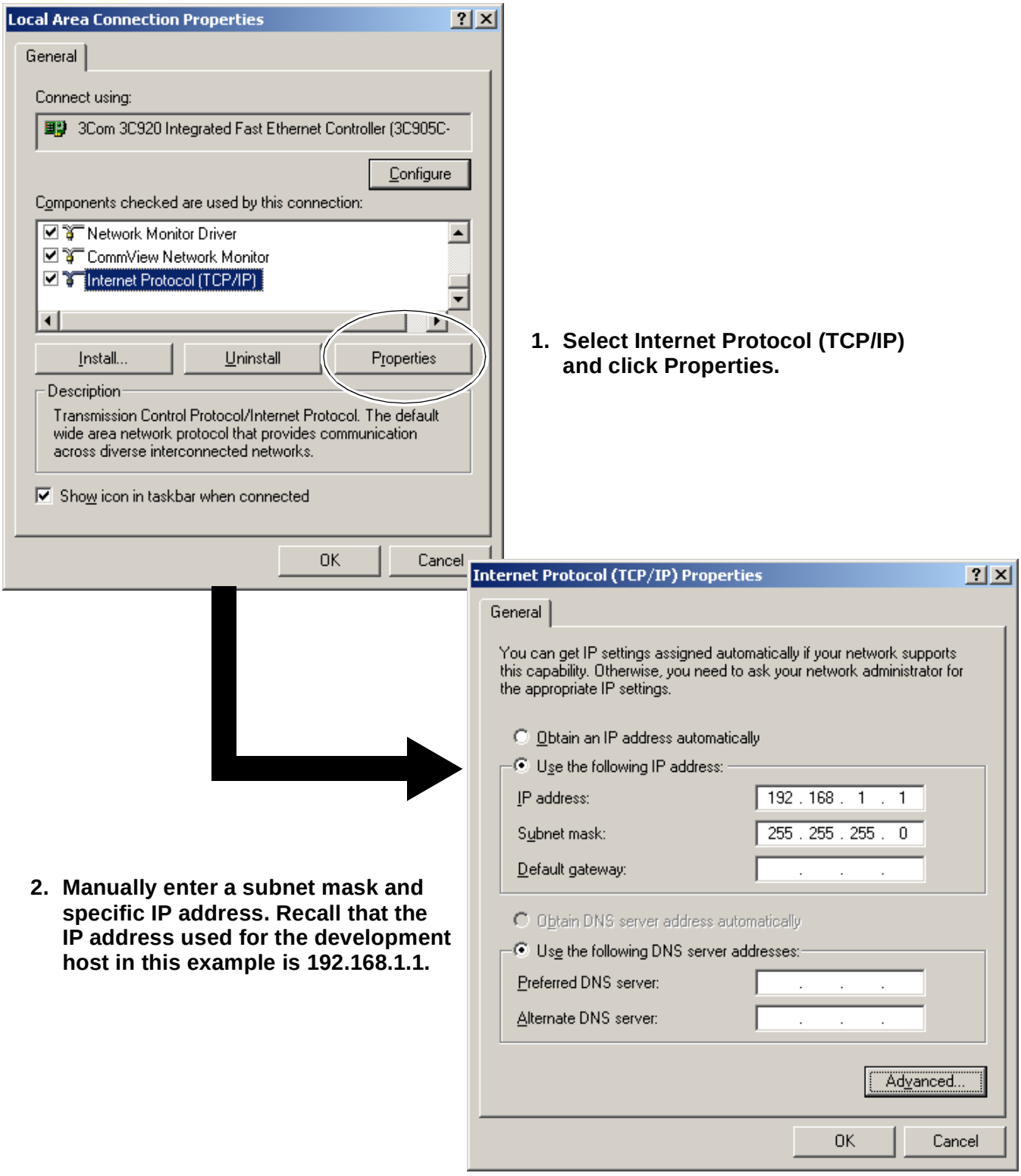
Figure 13. Network Settings Dialog Box

The network settings window shows all devices that can be used to form network connections. **Figure 13** 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 14**) 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.



1. **Select Internet Protocol (TCP/IP) and click Properties.**

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 14. 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 start-up
- 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 13**. Also, if the network is configured correctly, Windows may show the status of the link on the task bar as shown in **Figure 15**. The task bar shows the link as “Network Cable Unplugged.”

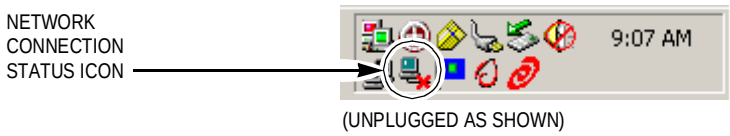


Figure 15. Task Bar Showing Link Status

Network Connection Cannot be Established

When 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 the **Configuring the IP Addresses** section for resolution and discussion).

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

This section provides an overview of a web server developed with CMX-MicroNet software. This simple web server was developed by CMX for distribution with the Technological Arts MC9S12E128 Ethernet reference design, as an S-record file. **Figure 16** shows the web server.

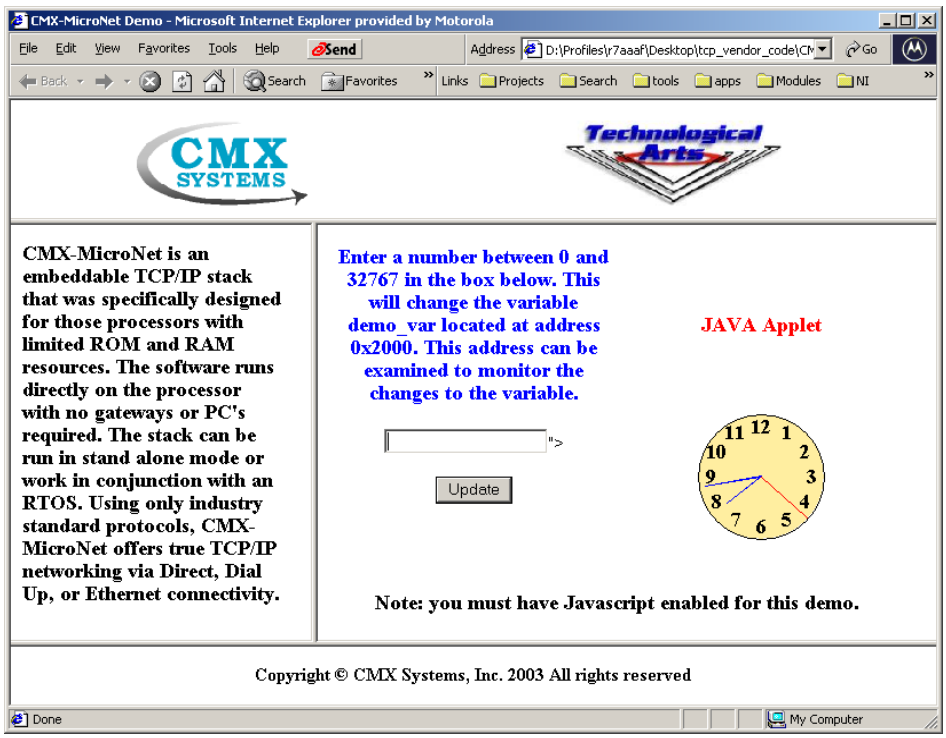


Figure 16. CMX-MicroNet Web Server

This section will overview the files that were modified during development of the CMX 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 16** 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.

HTML 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(), both the MC9S12E128 MCU and the LAN91C111 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 the**CMX-MicroNet TCP/IP Stack API** section 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 the**Configuring the MAC Hardware and IP Addresses** section 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 this file 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. 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

When editing *mnconfig.h*, recall that some protocols are dependent on each other. For instance, the HTTP protocol requires the TCP protocol. In addition, for CMX-MicroNet, either the UDP or TCP protocol must be enabled for compilation. **Figure 17** shows several popular network protocols and their dependence.

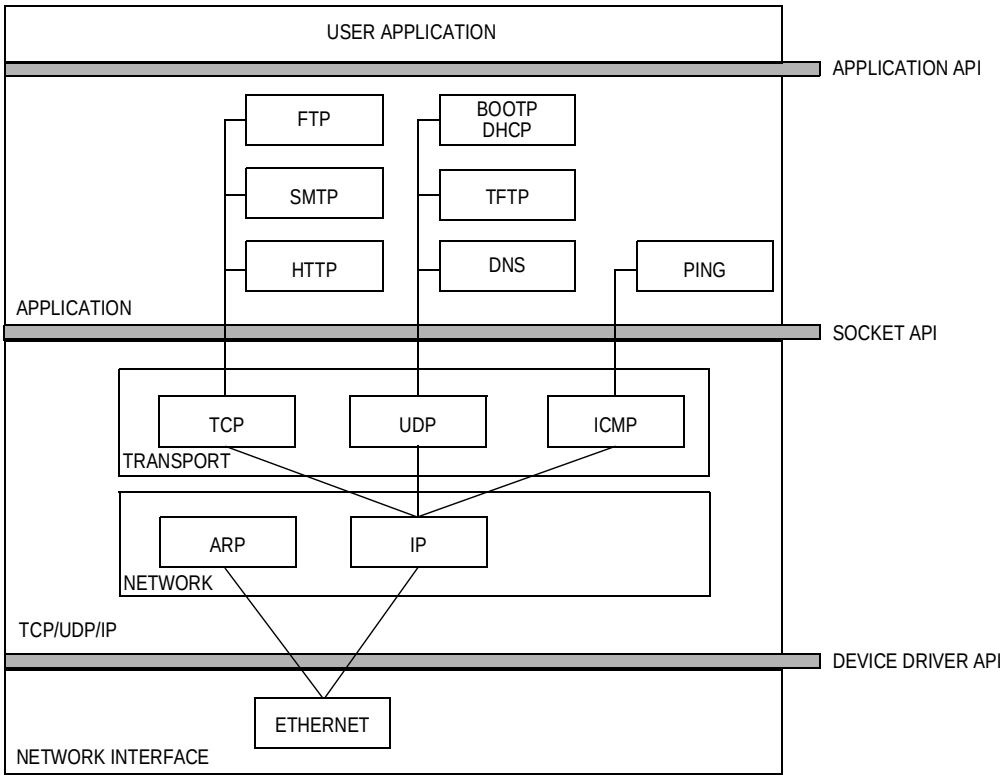


Figure 17. Popular Network Protocols and Their Dependence

```
/*********************************************************
Copyright (c) CMX Systems, Inc. 2002. All rights reserved
*********************************************************/

/* mn_env.h must be #included before this file in micronet.h */

#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

/* Sockets */
#define NUM_SOCKETS             6
#define SOCKET_WAIT_TICKS       600
#define RECV_BUFF_SIZE          2048
#define XMIT_BUFF_SIZE          1518
#define SOCKET_INACTIVITY_TIME  0 /*   7200 */

/* TCP/IP options */
#define TIME_TO_LIVE     64
#define TCP_WINDOW       1460
#define TCP_RESEND_TICKS 600
#define TCP_RESEND_TRYS  12
#define PING_GLEANING    0

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

/* ARP */
#define ARP              1
#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         1460

/* FTP */
#define FTP               0
#define FTP_SERVER        1
#define FTP_MAX_PARAM     24
#define FTP_BUFFER_LEN    1460
#define FTP_USER_LEN      10
#define FTP_PASSWORD_LEN  10
#define FTP_NUM_USERS     2
#define NEED_MEM_POOL     0
#define MEM_POOL_SIZE     4096

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

/* SMTP */
#define SMTP             0
#define SMTP_BUFFER_LEN  1460

/* Virtual File System */
#define VIRTUAL_FILE    1
#define NUM_VF_PAGES    12
#define VF_NAME_LEN     20
#define FUNC_NAME_LEN   20
#define NUM_POST_FUNCS  2
#define NUM_GET_FUNCS   2

#endif   /* ifndef MNCONFIG_H_INC */
```

hcs12e_91C111.c An excerpt of the source code for *hcs12e_91C11.c* is provided. The code shows two configurations a user may desire to modify. These include:

- DO_DEBUG — This option sets up the debug mode of the CMX-MicroNet stack. When this mode is configured, messages about the operation of the CMX-MicroNet stack are sent via the SCI.
- AUTO_NEGOTIATE — This option sets up smsc91C111_init() to initialize the LAN91C111 in auto-negotiation mode if asserted. If not asserted, the developer must manually set up the speed and duplex for the LAN91C111 using the full_duplex and speed_100 variables.

Excerpt from hcs12e_91C11.c:

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

/* driver for PC9S12E128 board with LAN91C111 Ethernet.

   For use with v2.15 and up only.
*/

#include "micronet.h"

#if (Ethernet)

#include "hc12e_91c111.h"

/* modifiable #defines */
#define DO_DEBUG        0     /* set to 1 to send debug info to UART */

#define EEPROM_PRESENT 0     /* set to 1 if EEPROM is being used */

/* set the following to 1 to have the chip auto-negotiate parameters, or set
   to 0 to use the full_duplex and speed_100 settings below.
*/
#define AUTO_NEGOTIATE     0

#if (!AUTO_NEGOTIATE)
int full_duplex = 1;             /* set to 1 for full duplex */
int speed_100 = 0;               /* set to 1 for 100Base-T, 0 for 10Base-T */
#endif                           /* AUTO_NEGOTIATE */

#define HARDWARE_PAD    1     /* set to 1 to enable hardware padding */
#define NUM_ALLOC_POLLS 500   /* times to poll if allocate completed */
#define NUM_TXSUC_POLLS 500   /* times to poll for TXSUC */
```

CMX 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 buffer 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 18 illustrates a simplified or partial TCP/IP stack implementation. This stack uses only the UDP protocol with TFTP and BOOTP as applications.

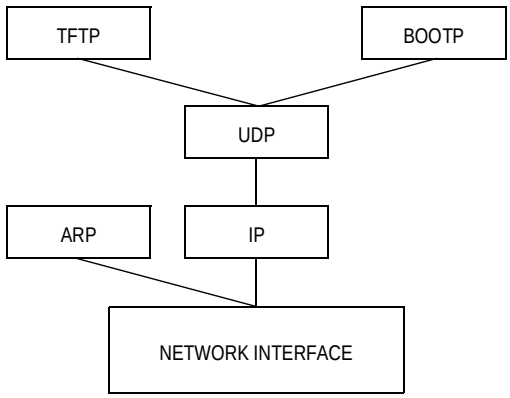


Figure 18. Partial Stack TCP/IP Stack Implementation

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.

Set Buffer to Appropriate Values

CMX-MicroNet uses a Tx and Rx buffer to operate on data coming from the LAN91C111. These buffers are allocated from RAM. RAM for buffers should be balanced with user application RAM. If large buffers are not required for the user application, set the buffer values so that the user application uses only the necessary RAM resources. In CMX-MicroNet, the buffers are set in the *mnconfig.h* file.

NOTES: *With the exception of mask set errata documents, if any other Motorola 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 were used to develop and debug this demo, Motorola 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:
Motorola Literature Distribution
P.O. Box 5405
Denver, Colorado 80217
1-800-521-6274 or 480-768-2130

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

ASIA/PACIFIC:
Motorola Semiconductors H.K. Ltd.
Silicon Harbour Centre
2 Dai King Street
Tai Po Industrial Estate
Tai Po, N.T., Hong Kong
852-26668334

HOME PAGE:
http://motorola.com/semiconductors

AN2624/D
Rev. 0