CMX-MicroNet Evaluation Version for the Motorola MC9S12E128

Getting Started Guide

TRADEMARKS:
MC9S12E128™ is a trademark of Motorola, Inc.
Metrowerks™ and Codewarrior™ are trademarks of Metrowerks Corporation
CMX™ is a trademark of CMX Systems, Inc.

The version of the Metrowerks tools used for CMX-MicroNet™ were:
Codewarrior Development Studio for Motorola HC12 3.0

Installation

The CMX-MicroNet software is distributed in the form of a setup.exe file.
If you received your software via email in a .zip file, the zip file is password-protected only to make it more likely to make it through corporate email firewalls. The zip file password is “cmx”.

Run setup.exe.
The default installation directory, c:\MICRONET, may be changed to another location.

Caution: We recommend installing the software in a root-level directory. The reason for this is that you may experience problems linking if the software is installed in too “deep” a directory. Some of the tools have a finite limit on the length of a directory/path specification, and if this limit is exceeded, then you will experience problems. This kind of issue has become much more common in recent years, with the advent of long file names.

When you are prompted for a password during installation, use “CMX-MICRONET” (case-sensitive).

What is installed:
Folders:

- Manual – We highly recommend reading the manual, mn303man.pdf.
- Demo_apps – This directory contains the source files for the projects.
- Demo_apps\web1_pages – Has the web pages used by the web1 application.
- Demo_apps\web2_pages – Has the web pages used by the web2 application.
- Mw_hc12\demos_E128\netlibe – contains the evaluation version of the netlibe library.
- Mw_hc12\demos_E128\tcp_app – Holds the TCP/IP echo client/server application.
- Mw_hc12\demos_E128\web1 – Holds a simple web server application.
- Mw_hc12\demos_E128\web2 – Holds a more complex web server application.
- Netlib – This directory normally hold the core CMX-MicroNet library. For the evaluation version only the CMX-MicroNet header files are here.
- PCtestprograms – These are programs, to run on a PC. They are used so that the CMX-MicroNet MC9S12E128 software has something to talk to. See the README.TXT file.
- Util – These are various utilities used in building the software. See the util.txt file.

Files:

- Tcp_app.c – example program for testing CMX-MicroNet TCP/IP echo client and server.
- Web1.c – example program showing a simple HTTP server.
- Web2.c – example program showing an HTTP server with server-side-includes, a POST function and a JAVA applet.
- Callback.c - user callback routines, described in the manual, for CMX-MicroNet. This is also where IP addresses etc are configured.
- Callback2.c - user callback routines modified for the web2 application. This is also where IP addresses etc are configured.
- install.log – Produced during the install, if you have an installation problem while running setup.exe, please email CMX tech support this file.
- license.txt – the software license
- unwise.exe – use this should you wish to uninstall the software

Please email CMX at the following email address to report bugs or problems with the CMX-MicroNet Evaluation Version: cmx@cmx.com

CMX-MicroNet Evaluation Version limitations

The CMX-MicroNet Evaluation Version has some limitations that the full CMX-MicroNet does not have.

- The evaluation version will only run for 30 minutes or send 1,000 TCP/IP and/or UDP/IP packets before locking up. The board may be reset to run for another 30 minutes or 1,000 packets. PING reply packets and ARP packets do not count towards the packet limit.
- The options included with the evaluation version are Ethernet, HTTP Server and TFTP.
- The Ethernet MAC address is fixed at 00-12-34-56-78-9A.
- The configuration defines in mnconfig.h are fixed and may not be changed. For example, in the full CMX-MicroNet up to 127 sockets can be open at a time, but in the evaluation version only five sockets may be open at one time. Some other restrictions are that the receive buffer is 900 bytes, the TCP_WINDOW, which is the amount of data that can be sent in a single packet, is 512 bytes, and six web pages, five GET functions and five POST functions may be added to the virtual file system. See the **Mnconfig.h** section below for a full list.
- The header files in the Netlib directory and the header files in the directories under the Mw_hc12\demos_E128\netlibe directory may not be changed.
- RTOS support is not included.

The following are limitations common to all versions of CMX-MicroNet.

- IP fragmentation is not supported. Note that this only affects TCP/IP packets. Protocols such as HTTP and FTP do not use fragmentation.
- IP options are ignored.
- ICMP only supports echo reply.
- TCP sends MSS option, received options are ignored.
- TCP respects other side's window, but uses a fixed window itself.
- Every TCP packet must get an ACK before the next one can be received. (No delayed ACKs).

Getting Started

➔ The included example programs and projects are very important in both verifying correct installation and configuration, but also in giving you a working piece of code.

➔Do not change any of the header files in the netlib directory. The netlibe library has been built using those files so changing any of them would cause a conflict between the library code and the application code.

Mnconfig.h

mnconfig.h is used to select the protocols used, number of sockets, sizes of transmit and receive buffers, etc. See the Configuration File section of the manual for details.

Here is what the mnconfig.h for the evaluation version looks like. For each of the #defines, there is an explanation in the manual which describes what the default settings are, what the setting does, etc.

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

The tcp_app application

This program can be configured to run as either a TCP echo client or TCP echo server by changing the following define in tcp_app.c:

```
#define SERVER_MODE         0    /* set to 1 if a server, or 0 if a client */
```

We recommend starting with the example set for TCP client mode, as shown above.

You also need to edit callback.c in the demo_apps directory to change the default network IP addresses. The gateway IP address and subnet mask should be set also. Application specific functions (callbacks) in this file can be changed, if required. If a gateway is not used (gateway IP address is set to 255.255.255.255) then the IP address of the PC and the IP address of the board must be set so they are on the same network. e.g. both are 192.168.2.xxx.

The program, when configured for CLIENT_MODE, will attempt to connect via TCP/IP to a TCP echo server (such as the included TCP echo server program tcp_svr.exe that runs on a PC) at the destination IP location indicated in callback.c and using the echo service port 7. It sends data continuously to this address and then receives it back from the echo server.

You could then do the opposite, use the PC tcp_cli.exe program, and set the example to run as a server. In this case, the PC will send data to the example program running on the MC9S12E128, which will then echo it back to the PC. When using tcp_cli.exe you must supply a parameter that is the same as the IP address set in the ip_src_addr array in callback.c. e.g.

```
Tcp_cli 192.168.2.3
```

Tcp_cli.exe and Tcp_svr.exe both display the data they receive on the screen.

The web1 application

The web1 application demonstrates how to use the HTTP server to serve up a simple web page. The web page, index.htm, and its two graphics files are in the demo_apps\web1_pages directory. The HTML2C utility has been used to convert those files into .c and .h files that are included in the project. Note that the main web page must be called index.htm or index.html.

If server-side-includes, POST functions and JAVA applets are not used, a web server application can be created in just a few steps.

- Convert web pages to .c and .h files using the HTML2C utility.
- #include the created .h files in your application after #include “micronet.h”
- In the main function call mn_init() before calling any other CMX-MicroNet functions.
- Add the web pages to the virtual file system with the mn_vf_set_entry() function call and the parameters defined in the .h files created by HTML2C.
- Call the mn_server() function. This function normally does not return.

See the **Using the included Metrowerks projects** section for more details on using the web1 application.

The web2 application

The web2 application serves up a web page with two frames, two server-side-includes, a form and a JAVA applet. If your OS does not have a JAVA virtual machine go to http://java.sun.com/getjava and download the JAVA Runtime Environment (JRE) for your OS. After the JRE is installed go to Control Panel, Java Plug-in, Cache and remove the check from Enable Caching and click Apply.

Server-side-includes are a way of inserting dynamic data into a web page. A special tag is placed into the web page specifying a GET function to be called by CMX-MicroNet. A GET function must be placed into the virtual file system with a call to function mn_gf_set_entry(). This user-defined function passes the data to be placed into the web page to the HTTP server, which then replaces the tag with the passed data. For example in bot.htm there is the tag:

```
<param name=Tick value="<!--#exec cgi="getTickVal"-->">
```

When the HTTP server sees the "<!--#exec cgi= string it looks for a function name inside the following quotes. It then looks up the function name in the virtual file system and runs the associated function, which in this case is get_tick_func. A GET function must take a pointer to a pointer to a buffer as a parameter and return the number of bytes placed in the buffer as a word16 variable. See web2.c and the CMX-MicroNet manual for details.

Forms in web pages are handled through POST functions. The ACTION attribute of the form is set to the name of a user-defined POST function. A GET function must be placed into the virtual file system with a call to function mn_pf_set_entry(). When the submit button of the form is clicked the function associated with the POST function name is executed. A POST function is passed a pointer to the socket associated with the web page. The mn_http_find_value function can be called to get the value(s) of the variable(s) in the POST request. This function either the mn_http_set_message function must be called to send a message back to the browser or the mn_http_set_file called to send a web page back to the browser. In the web2 example it looks for a variable called display and if found places the passed value in the msg_buff array. If the msg_buff array was successfully updated a HTTPStatus204 message is sent. This tells the web browser that the POST was successful but that no web page will be returned. See web2.c, bot.htm and the CMX-MicroNet manual for details.

The web server can also serve up JAVA applets. The applet .class files are converted to .c and .h files using HTML2C and added to the virtual file system the same as other web pages. A powerful feature is the ability of JAVA applets to establish a TCP connection with CMX-MicroNet thus allowing immediate bi-directional communication between the applet and the board. In the web2 example a socket is opened up for listening on port 2000 before the HTTP server is started. The JAVA applet opens up a TCP connection at startup and then listens for data coming from the board. Function mn_app_server_idle() in callback2.c is called whenever the web server is not busy processing packets. . In the web2 example this function has been modified to make sure there is a socket on port 2000 available to listen for incoming connections and every five seconds the system timer tick value is sent to all sockets with a destination port of 2000. Note that multiple JAVA applets may connect to the board at the same time. See web2demo.java, web2.c and callback2.c for more details.

See the **Using the included Metrowerks projects** section for more details on using the web2 application.

Using the included Metrowerks projects

The following project files are provided in the Mw_hc12\demos_E128 directory.

TCP echo client or server	Tcp_app\tcp_app.mcp
Simple web server	Web1\web1.mcp
More advanced web server	Web2\web2.mcp

➔ Open the projects using Codewarrior Development Studio for Motorola HC12 3.0.

➔ Before running the project, make sure callback.c has the desired IP network settings.

If you change one of the web pages in the web server examples then you must run the HTML2C utility found in the util directory to create new .c and .h files. For example if index.htm is modified you would run:

```
Html2c index.htm
```

That will create index.c and index.h. See the Virtual File System section of the CMX-MicroNet manual for more information on using the Virtual File System.

To access the web pages using one of the HTTP server examples, in the browser's address box enter http:// followed by the board's IP address defined in callback.c. e.g.

http://192.168.2.3

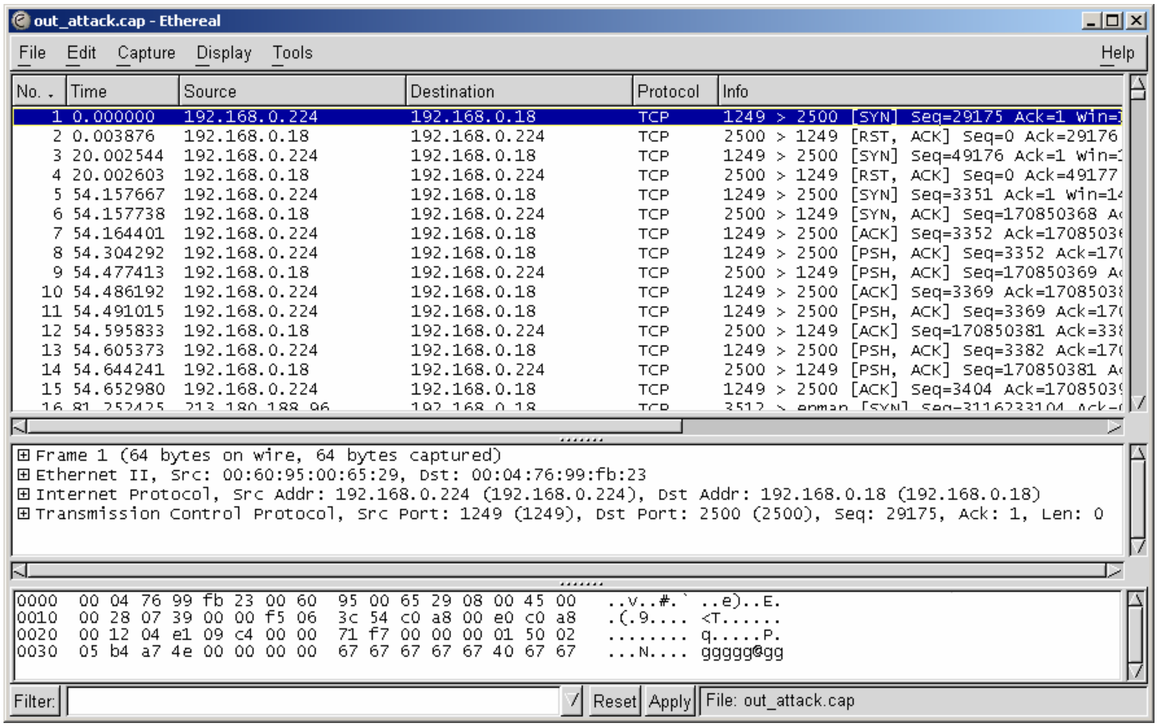
Debugging

Besides using the included PC-side TCP/UDP client/server test programs, we highly recommend the use of a packet sniffer. These allow you to see all transmitted frames and see exactly what is going on. Some of the freeware ones, like Ethereal, are surprisingly good.

FAQ - http://www.robertgraham.com/pubs/sniffing-faq.html

Freeware Packet Sniffers for Windows
AnalogX PacketMon - www.analogx.com
Anasil - www.sniff-tech.com
CommView - www.tamosoft.com
Ethereal - www.ethereal.com
Sniff'em - www.sniff-em.com

Commercial
Klos Technologies' SerialView, PacketView www.klos.com
Windows Packet sniffing library for C#, C++, VB - http://www.packet-sniffing.com


out_attack.cap - Ethereal
File Edit Capture Display Tools Help
Filter: Reset Apply File: out_attack.cap

```
No. . Time       Source          Destination     Protocol Info
    1 0.000000   192.168.0.224   192.168.0.18    TCP      1249 > 2500 [SYN] Seq=29175 Ack=1 Win=1
    2 0.003876   192.168.0.18    192.168.0.224   TCP      2500 > 1249 [RST, ACK] Seq=0 Ack=29176
    3 20.002544  192.168.0.224   192.168.0.18    TCP      1249 > 2500 [SYN] Seq=49176 Ack=1 Win=1
    4 20.002603  192.168.0.18    192.168.0.224   TCP      2500 > 1249 [RST, ACK] Seq=0 Ack=49177
    5 54.157667  192.168.0.224   192.168.0.18    TCP      1249 > 2500 [SYN] Seq=3351 Ack=1 Win=14
    6 54.157738  192.168.0.18    192.168.0.224   TCP      2500 > 1249 [SYN, ACK] Seq=170850368 Ac
    7 54.164401  192.168.0.224   192.168.0.18    TCP      1249 > 2500 [ACK] Seq=3352 Ack=1708503
    8 54.304292  192.168.0.224   192.168.0.18    TCP      1249 > 2500 [PSH, ACK] Seq=3352 Ack=170
    9 54.477413  192.168.0.18    192.168.0.224   TCP      2500 > 1249 [PSH, ACK] Seq=170850369 Ac
   10 54.486192  192.168.0.224   192.168.0.18    TCP      1249 > 2500 [ACK] Seq=3369 Ack=1708503
   11 54.491015  192.168.0.224   192.168.0.18    TCP      1249 > 2500 [PSH, ACK] Seq=3369 Ack=170
   12 54.595833  192.168.0.18    192.168.0.224   TCP      2500 > 1249 [ACK] Seq=170850381 Ack=33
   13 54.605373  192.168.0.224   192.168.0.18    TCP      1249 > 2500 [PSH, ACK] Seq=3382 Ack=170
   14 54.644241  192.168.0.18    192.168.0.224   TCP      2500 > 1249 [PSH, ACK] Seq=170850381 Ac
   15 54.652980  192.168.0.224   192.168.0.18    TCP      1249 > 2500 [ACK] Seq=3404 Ack=1708503
   16 81.252425  213.180.188.96  192.168.0.18    TCP      3512 > epmap [SYN] Seq=3116233104 Ack=

⊞ Frame 1 (64 bytes on wire, 64 bytes captured)
⊞ Ethernet II, Src: 00:60:95:00:65:29, Dst: 00:04:76:99:fb:23
⊞ Internet Protocol, Src Addr: 192.168.0.224 (192.168.0.224), Dst Addr: 192.168.0.18 (192.168.0.18)
⊞ Transmission Control Protocol, Src Port: 1249 (1249), Dst Port: 2500 (2500), Seq: 29175, Ack: 1, Len: 0

0000  00 04 76 99 fb 23 00 60  95 00 65 29 08 00 45 00   ..v..#.` ..e)..E.
0010  00 28 07 39 00 00 f5 06  3c 54 c0 a8 00 e0 c0 a8   .(.9.... <T......
0020  00 12 04 e1 09 c4 00 00  71 f7 00 00 00 01 50 02   ........ q.....P.
0030  05 b4 a7 4e 00 00 00 00  67 67 67 67 67 40 67 67   ...N.... ggggg@gg
```

Figure 1 Ethereal freeware packet sniffer