NEC

Application Note

CMX-Micronet

For 78K0 Family

Application Programmers Interface

Document No. U16225EE1V0AN00
Date published: April 2002

Table of Contents

List of Figures 4
List of Tables 5
Introduction 6

1 Overview 8
1.1. The 78K0 device μPD78F0066 8
1.2. CMX Micronet 9

2 TCP/IP stack theory including application layers 10
2.1. The TCP/IP Stack 10
2.1.1. Internet Protocol (IP) 11
2.1.2. ICMP 11
2.1.3. UDP 11
2.1.4. TCP 11
2.2. Application Layer 11
2.2.1. DNS 11
2.2.2. TFTP 11
2.2.3. PING 11
2.2.4. Application Ports 12
2.2.4.1. HTTP Server 12
2.2.4.2. FTP Server 12
2.2.4.3. SMTP client 12
2.2.4.4. BOOTP 12
2.2.4.5. DHCP 12
2.3. Physical Layer Protocols 12
2.3.1. Serial 12
2.3.1.1. PPP 12
2.3.1.2. SLIP 12
2.3.2. Ethernet 13
2.3.2.1. ARP 13
2.3.2.2. RARP 13
2.4. User Application 13
2.4.1. HTML 13
2.4.1.1. Server Side Includes(SSI) 13
2.4.1.2. POST Functions 13
2.4.2. XML 13
2.4.3. JAVA 13
2.5. Peer Type 14
2.5.1. Client 14
2.5.2. Server 14

3 System Configuration 15
3.1. Memory Map 17
3.2. RAM 18
3.2.1. Link Layer 18
3.2.2. CMX Micronet Stack 18
3.2.3. Virtual Files 21
3.2.4. User Program available RAM 21

4 Hardware and Peripherals 23
4.1 PD78F0066 Peripheral and I/O Assignments 23
4.2. Timer Functions 23
4.3. Interrupts 23

5 Description of Software 24
5.1. Software Location Setup 24
5.2. Context Diagram 25
5.3. Interaction with CMX-Micronet 25
5.4. State Diagram for Example.c 27
5.5. Software Discriptions 28
5.5.1. CMX Micronet Setup 28
5.5.1.1. Switch Settings 30
5.5.1.2. User Application Initialisation 30
5.5.1.3. CMX-Micronet Initialisation 30
5.5.1.4. Virtual File Setup 30
5.5.1.5. POST Function Setup 30
5.5.1.6. Server Side Include Setup 31
5.5.2. PC Connection Started 31
5.5.3. HTTP Server 31
5.5.4. PPP Close 31
5.5.5. MODEM Close 32

6 Application Programmers Interface (API) 33
6.1. Initialisation 33
6.2. HTTP 33
6.3. TCP 34
6.4. IP 35
6.5. PPP 35
6.6. Callback 35
6.7. MODEM 36
6.8. Virtual File System 37
6.8.1. HTML 37
6.8.1.1. Server Side Includes(SSI) 38
6.8.1.2. POST Functions 39

7 Example Code 40
7.1. API Build 40
7.1.1. Example.c 40
7.1.2. Callback.c 49
7.1.3. nec_hw.c 52
7.1.4. init.c 56
7.2. Examples 58
7.2.1. Electronic Dice and A/D Input 58
7.2.1.1. Example.c 59
7.2.1.2. Callback.c 69
7.2.1.3. Nec_w.c 72
7.2.1.4. Init.c 76
7.2.1.5. Conversion.c 79
7.2.1.6. AtoD.c 81
7.2.1.7. Dice.c 83
7.2.1.8. Interrupt.c 86

Index 89

List of Figures

Figure 2-1 TCP/IP Protocol Stack Layers....10
Figure 3-1 Functional Blocks within CMX-Micronet....16
Figure 3-2 Memory Map....17
Figure 3-3 RAM Usage....18
Figure 5-1 Directory Structure....24
Figure 5-2 Context Diagram....25
Figure 5-3 User Application Active Only When Needed....26
Figure 5-4 User Application Active All The Time....26
Figure 5-5 User Application Using Both Types Of Interaction....27
Figure 5-6 State Diagram for Example.c....28
Figure 5-7 CMX-Micronet Setup Flow Diagram....29
Figure 5-8 Simplified State Diagram for HTTP Server....31

List of Tables

Table 3-1 Array RAM usage 20
Table 4-1 I/O Assignment 23

Introduction

Target Readers

This application note is intended for users who understand the functions of the 78K0 family and who design application systems using these microcontrollers. The software was developed on the 78K -Test It! kit, but can be provided in source code format to be used on any 78K0 microcontroller.

Purpose

The purpose of this application note is to help users understand the use and composition of CMX Micronet functions, and to understand how users can interact with a protocol stack and various application layers associated with connectivity. The system example uses the 78K – Test It! application programs to demonstrate the use of CMX Micronet. Application features include Web page development interaction, POST, SSI and the use of the application layer options included within the CMX Micronet structure.

Organization

This application note is divided into the following sections:

- Overview of μPD78F0066 and CMX-Micronet
- TCP/IP stack theory including application layers
- System Configuration used for this application note
- Hardware and peripherals used for this application
- Description of software
- Application Programmers Interface(API)
- Example Code

How to use this Manual

It is assumed that the reader of this application note has general knowledge in the fields of electronic engineering, logic circuits, and microcomputers. For details of hardware and instruction functions (especially register functions, setting methods, etc.) see the **μPD76F0066 Hardware User's Manual**.

Conventions

Data significance: Higher digits on the left and lower digits on the right.

Active low representation:xxx (overscore over pin or signal name).

Memory map address: Higher address on the top and lower address on the bottom.

Note: Explanation of (Note) in the text

Caution: Information requiring particular attention

Remark: Supplementary explanation to the text

Numeric notation :

Binary	xxxx or xxxB
Decimal	xxxx
Hexadecimal	xxxxH or 0x xxxx

Prefixes representing powers of 2 (address space, memory capacity)

K (kilo) : $2^{10} = 1024$
M (mega) : $2^{20} = 1024^2 = 1,048,576$
G (giga) : $2^{30} = 1024^3 = 1,073,741,824$

Data Type:

Word: 32 bits
Halfword:16 bits
Byte: 8 bits

Related Documents: The related documents indicated in this publication may include preliminary versions. However, preliminary versions are not marked as such.

Documents related to μPD78F0066

Document Name	Document No.
μPD78F0066 User's Manual	U13420EJ2V0UM00
μPD78F0066 Data Sheet	U13419EJ1V0PM00

Documents related to development tools (User's Manuals)

Document Name	Document No.
78K-Test It! Starter Kit	78K-TESTIT090V10

Documents related to CMX Systems

Document Name	Document No.
CMX-Micronet (Not provided in evaluation version)	mn213man

1 Overview

This document outlines specifics related to the device µPD78F0066 and its use with the Internet protocol stack called CMX-Micronet from CMX Systems. It is the intention, however, to use the same basic architecture of this device and relate this to any 78K0 microcontroller available. Therefore the hardware required to implement CMX-Micronet is common to all 78K0 devices. The only exception to this is the amount of RAM available on other 78K0 devices The µPD78F0066 has the largest amount of RAM available in the 78K0 family.

1.1. The 78K0 device µPD78F0066

Features

	Internal Memory		
	Type Program Memory		Data Memory
Part Number (Mask ROM/Flash memory)	High-Speed RAM	Expansion RAM	Buffer RAM
µPD780065 40 Kbytes	1024 bytes	4096 bytes	32 bytes
µPD78F0066 48 Kbytes[1]			

- External Memory Expansion Space: 64 Kbytes
- Minimum instruction execution time changeable from high speed (0.24 µs: @ 8.38-MHz operation with main system clock) to ultra-low speed (122 µs: @ 32.768-kHz operation with subsystem clock)
- Instruction set suited to system control
- Bit manipulation possible in all address spaces
- Multiply and divide instructions
- Sixty I/O ports
- 8-bit resolution A/D converter: 8 channels
- Serial interface: 4 channels
- 3-wire serial I/O mode (provided with automatic transmission/reception function): 1 channel
- 3-wire serial I/O mode: 1 channel
- 2-wire serial I/O mode: 1 channel
- UART mode: 1 channel
- Timer: Five channels
- 16-bit timer/event counter: 1 channel
- 8-bit timer/event counter: 2 channels
- Watch timer: 1 channel
- Watchdog timer: 1 channel
- Vectored interrupts: 20
- Two types of on-chip clock oscillators (main system clock and subsystem clock)
- Power supply voltage: VDD = 2.7 to 5.5 V

Note 1: The capacity of internal flash memory can be changed by means of the memory size switching register (IMS).

1.2. CMX Micronet

The CMX-Micronet TCP/IP Protocol Stack has been designed to work on processors with small amounts of ROM and RAM. This low ROM and RAM usage make it an ideal TCP/IP stack to implement on an 8-bit processor such as the 78K0 family and also on the low end 32-bit RISC V850 family in single chip mode with no external memory.

The CMX-Micronet Stack is supplied as C source code. But for evaluation purposes an object only time limited version is supplied with this system. An outline of the features enabled in this version are described in Chapter 3.

Features of CMX-Micronet TCP/IP Stack

User Application
- HTML Web pages
  - Server Side Includes (SSI)
  - POST functions
- Virtual file System

Application Layer
- SMTP client
- HTTP server
- FTP server
- TFTP
- BOOTP
- DHCP
- PING

Transport Layer
- TCP
- UDP

Network Layer
- IP
- ICMP

Link Layer
- ARP
- PPP
- PAP
- SLIP
- Ethernet Driver
- UART driver

MODEM, or direct Cable connection via RS232
Support for 16 sockets using UDP or TCP
All 16 sockets can be either Ethernet, PPP, or SLIP, but not mixed

Due to the low ROM and RAM footprint to Micronet there are certain departures made from the RFC standards to implement this.

Restrictions:
- PPP can negotiate IP addresses, but all other options are ignored.
- TCP ignores all options
- Micronet does not handle fragmented packets
- For evaluation purposes the CMX-Micronet stack is time limited

However these restrictions do not affect the development of an application requiring Internet protocol based connectivity.

2 TCP/IP stack theory including application layers

This chapter is designed to highlight the basic protocols available within the CMX-Micronet TCP/IP stack and to give a brief overview of each. It is assumed that the theory behind how the TCP/IP stack operates and the associated application layer protocols are not required within this document. There are many respectable books available on the subject of TCP/IP and so a detailed description is not needed.

Figure 2-1 TCP/IP Protocol Stack Layers

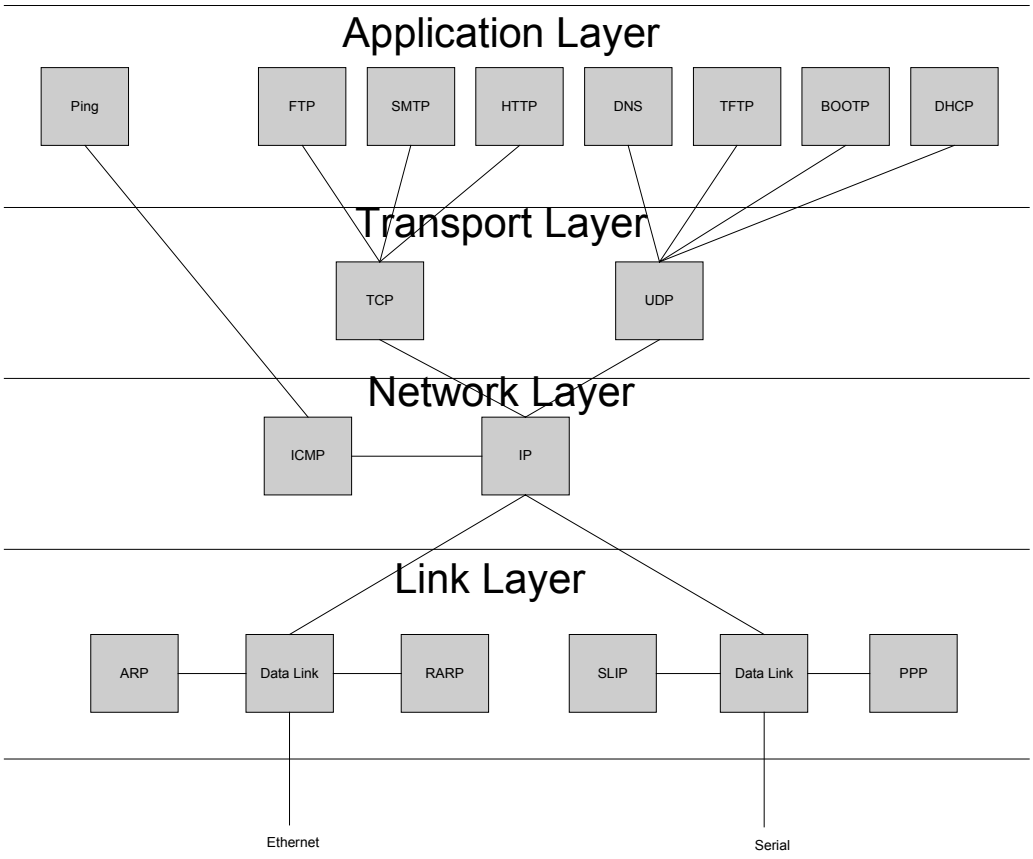


Figure 2-1 shows the different protocol names and layers within a TCP/IP stack. There are additional applications layers available, however for reference the protocols mentioned in figure 2-1 are touched upon within this chapter.

2.1. The TCP/IP Stack

The TCP/IP stack is a family of protocols specifically designed to aid in the movement of information between diverse systems. The stack is also designed to be independent of the media over which the information is to be transferred. Because of the widespread adoption of the protocol stack and the uptake of the Internet this is an ideal protocol family to use. The negative side of using the TCP/IP stack is that the stack was designed for desktop systems where memory usage is not a critical element, in an embedded system we have to be very careful with memory usage and most of the optimisations undertaken are to reduce memory usage. The TCP/IP stack consists of the following protocols

TCP Transport Control Protocol
UDP User Datagram Protocol
IP Internet Protocol
ICMP Internet Control Message Protocol

The above provide the main core of the TCP/IP protocol, the following will provide an overview of each of the elements.

2.1.1. Internet Protocol (IP)

Internet Protocol is designed to interconnect packet switch communication networks to form an Internet through which to pass data. The IP protocol provides four main functions

Basic unit for transfer
Protocol Addressing
Routing of Datagrams or Packets
Fragmentation of datagrams

As can be seen the IP is responsible for the addressing, each device on an Internet or Intranet must have a unique IP address, on the current version of CMX-Micronet Ipv4 addressing is supported.

2.1.2. ICMP

Internet Control Message Protocol is a companion protocol to the IP and must be present.The purpose of ICMP is to provide feedback about problems in the communications system.

2.1.3. UDP

User Datagram Protocol is a simple connectionless protocol, i.e. there is no guarantee of delivery. The major uses for this protocol are to support the following application level protocols.
DNS
TFTP
BOOTP

2.1.4. TCP

TCP is the transport layer protocol, a connection orientated protocol, i.e every packet must be acknowledged. TCP is also a full duplex protocol and data in both directions must be opened and closed independantly. TCP resides above the IP protocol and provides the interface to the various application ports, data is passed to and from the IP stack via the application ports. Delivery integrity is maintained via the use of sequence and acknowledge numbers. TCP is used to support the following application level protocols
PAP
FTP
SMTP
HTTP

2.2. Application Layer

2.2.1. DNS

Domain Name System is a simple map between host names and IP addresses. It is also used to provide mail routing information. The DNS provides the protocol that allows clients and servers to communicate with each other.

2.2.2. TFTP

Trivial File Transfer Protocol is designed to be used when initialising hardware from a diskless system. This is intended transfer small files using UDP and therefore fit into the internal ROM of the embedded device. This application is typically used in conjunction with BOOTP.

2.2.3. PING

The ping application program is designed to test whether another host connected to the system is reachable. This program sends an ICMP echo request message to host, expecting an ICMP echo reply to be returned.

2.2.4. Application Ports

The main area's of concern during the development of an embedded Internet application are the interfaces to the application ports and this will be the major focus of the API, the following is a brief explanation of the application ports used.

2.2.4.1. HTTP Server

The HTTP (Web) server is the application port for embedded Web applications. The port number for HTTP is always port 80 . The Webserver will provide the embedded HTML pages to the browser upon request from the browser. Acquired data from the user application (temperature measurement etc) can be dynamically inserted in to the Web page for remote monitoring. The Webserver can also call up user defined functions as long as they are included in a special format within the embedded HTML page.

2.2.4.2. FTP Server

The FTP (File Transfer Protocol) server allows for the transfer of files to and from an embedded application. The application port used is always port 21. This maybe used to transfer files that are stored on the application, i.e. log files etc.

2.2.4.3. SMTP client

SMTP (Simple Mail Transport Protocol) allows for the email notification of events to occur. SMTP always uses application port 25 . Email notification is ideal for alarm events, etc. SMTP is the standard Internet mail protocol, and allows for example a notification to be sent to an operator when an event occurs that requires user intervention.

2.2.4.4. BOOTP

Bootstrap Protocol is used when the embedded system has initialised with no knowledge of its IP address when using Ethernet. The IP address is detirmined by using RARP when bootstrap is enabled. It usually is used in conjunction with TFTP. BOOTP usually uses port 67 for server and 68 for client.

2.2.4.5. DHCP

Dynamic Host Configuration Protocol is an alternative to BOOTP and also its replacement. BOOTP has limited flexibility and therefore DHCP improves on this.

2.3. Physical Layer Protocols

There are several physical layer options available, but initially we will focus on the serial interface for 78K0 and later the Ethernet interface for V850 providing the physical layer connection. To use the serial interface we must use an appropriate physical layer protocol, in this case the protocol can be PPP (Point to Point Protocol), or SLIP (Serial Line Internet Protocol). For Ethernet ARP(Address Resolution Protocol) and its counterpart RARP (Reverse Address Resolution Protocol) are required to implement the physical layer.

2.3.1. Serial

2.3.1.1. PPP

Point To Point Protocol establishes the connection and negotiates the applicable network protocol (IP in the embedded Internet case). During the PPP negotiation phase the IP addressing is determined.

2.3.1.1.1. PAP

Password Authentication Protocol is a simple protocol for requesting a username and password to connect to the host. This user name and password is compared to those stored on the host and if a match then the user has access.

2.3.1.2. SLIP

Serial Line Internet Protocol is an alternative to PPP. Many PC's and MODEMs still use this protocol and so it is still provided.

2.3.2. Ethernet

2.3.2.1. ARP

Address Resolution Protocol provides a mapping between two different forms of addresses. The 32-bit IP address to the 48-bit Ethernet Address (MAC address). This mapping is held in the form of a table of the locally connected devices on the network.

2.3.2.2. RARP

Reverse Address Resolution Protocol works in the reverse direction to ARP in that it contains a mapping of the 48-bit Ethernet Address to the 32-bit IP address.

2.4. User Application

The User Applications contain customer specific applications. These specific applications interact through the various application ports and application programs within the TCP/IP protocol stack. The file System contains a list of the files available to the user both in ROM and RAM

2.4.1. HTML

HyperText Markup Language is a markup language which is a text only language. It is not a high level language such as C, but instead of being compiled and executed, it is read, or interpreted by what is termed a user agent. The user agent is commonly known as a Web browser. HTML uses HTTP to transfer information independent of the type of computer, or language and allows the of exchange of information across different networks.

2.4.1.1. Server Side Includes(SSI)

Server Side Includes, sometimes known as a GET function, are a feature of the HTML language. It allows the Web server to skim the contents of the HTML file looking for server based commands. When it finds one it performs the requested action, and places the results of the action in the place of the SSI statement. This is ideal for an embedded application because it allows the results of some action to be viewed within a Web browser. Such applications allow the user implement such things as remote diagnostics, etc.

2.4.1.2. POST Functions

A POST function is another HTML feature. It is used when form data is sent from the Web browser to the embedded Web server. The function allows the transfer of of data back to the embedded Web server which the embedded Web server can at apon. An application of this type would use a POST function to change variables within the Web server which can be acted apon by a user application.

2.4.2. XML

Extensible Markup Language can be used in conjunction with HTML to allow the creation and management of data which can be kept on the embedded Webserver. XML enables data processing that HTML is unable to do. Typical examples of the use of XML include the efficient use of database management, and so on an embedded Webserver a very simple database, can be implemented such as a datalogger when the data needs to be addressed in a simple way using a Web browser.

2.4.3. JAVA

JAVA applets allow JAVA enabled Web browsers to compile from within a HTML document JAVA code to run on a JAVA Runtime Engine (JRE). The clear advantage of JAVA over HTML then becomes clear when the Web browser running a JAVA applet can come to an end result after running such an applet without sending any information, or request any additional information from the embedded Web server.

An example of this is a JAVA applet to plot x and y co-ordinates for a temperature sensor over time. The JAVA applet is loaded initially from the embedded Web server. The Web browser runs the applet and only requests from the Web browser a new data sample for the temperature sensor, without the need to request the time. The HTML document does not need to be re-loaded to plot temperature over time and display the new information.

2.5. Peer Type

The term client and server are used throughout the protocol world and are therefore used throughout this document. It is important to understand why they are used, so we have outlined these terms. The CMX-Mictonet protocol stack makes reference to these within the application layer.

2.5.1. Client

Client simply means the requester of a service; ie this means that the client must initiate some task to get a response.

SMTP client application initiate the sending of an email to another location external to the device running CMX-Micronet.

2.5.2. Server

Server is the opposite to client in that a server must wait for a request from the client. The server responds to the task give by the client.

Therefore HTTP Server means that this application must wait for a resonse from the client which is external to the device running CMX-Micronet.

3 System Configuration

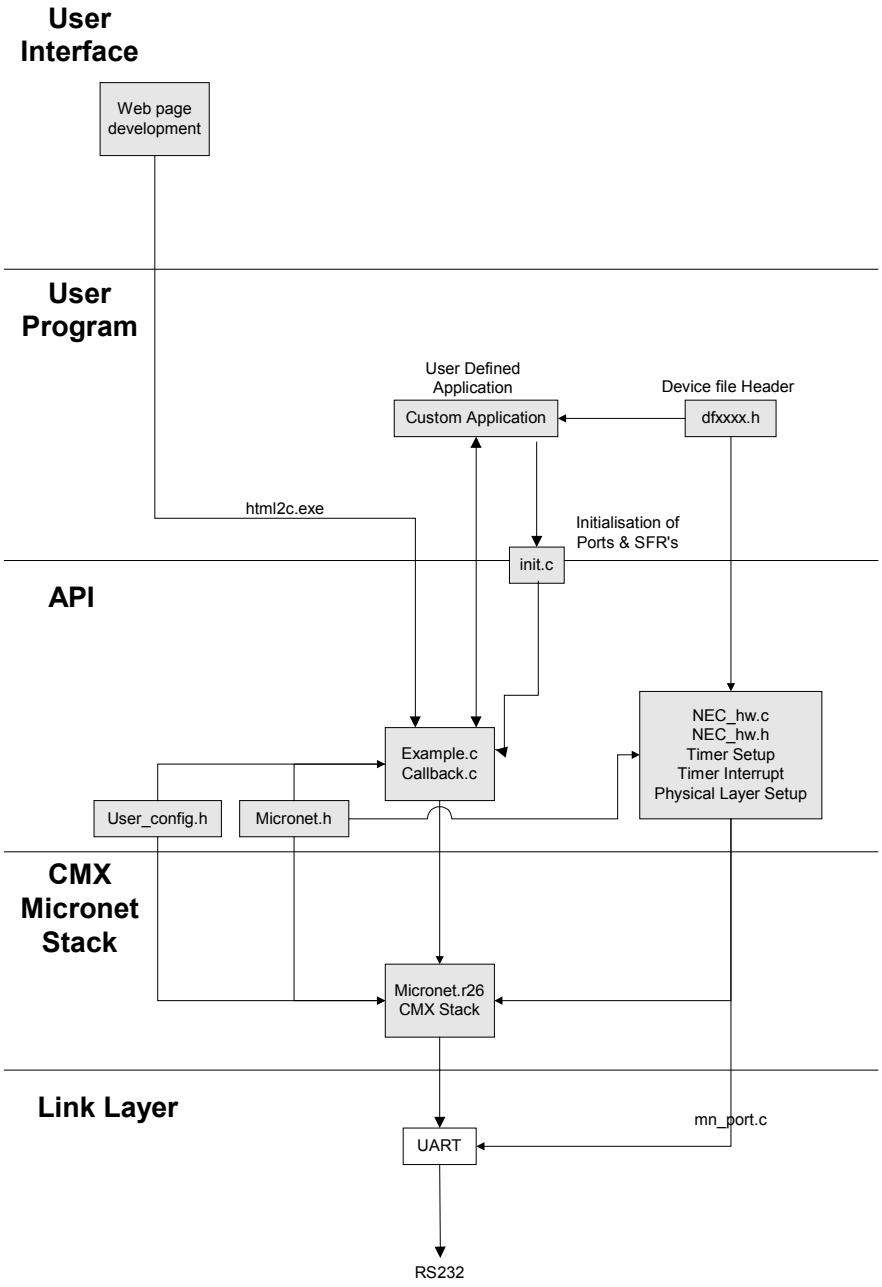
The following was developed on the 78K – Test It! evaluation board. However this document is designed to be used irrespective of the 78K0 device used.

For evaluation purposes the following options have been enabled within the object code time limited version to simplify the approval process for the CMX-Micronet protocol stack. All other options listed at the front of this document have been tested and will be available once evaluation has been completed and C source code provided.

User Application		
	HTML Web pages	
	Server Side Includes (SSI)	
	POST functions	
	Virtual file System	
Application Layer		
	HTTP server	
	PING	
Transport Layer		
	TCP	6 sockets
Network Layer		
	IP	
	ICMP	
Link Layer		
	PPP	
	UART driver	
Direct Cable connection via RS232		

Figure 3-1 shows a block diagram of the software functional blocks within the evaluation system.

Figure 3-1 Functional Blocks within CMX-Micronet



The CMX-Micronet stack is supplied in object code format with a time limitation of approximately 30min. While the layer above(API layer) is supplied as source code for user interaction. Examples of user program interaction with the API layer are contained in Chapter 7. While in Chapter 6 enters into how to use the API.

Headers:

User_config.h	-	contains compiler tests to check if config.h has been set up correctly
Micronet.h	-	Links all header information. Must be added to all user application code.
NEC_hw.h	-	Contains definitions used for peripheral setup.
DF0066.h	-	Device file header for μPD78F0066

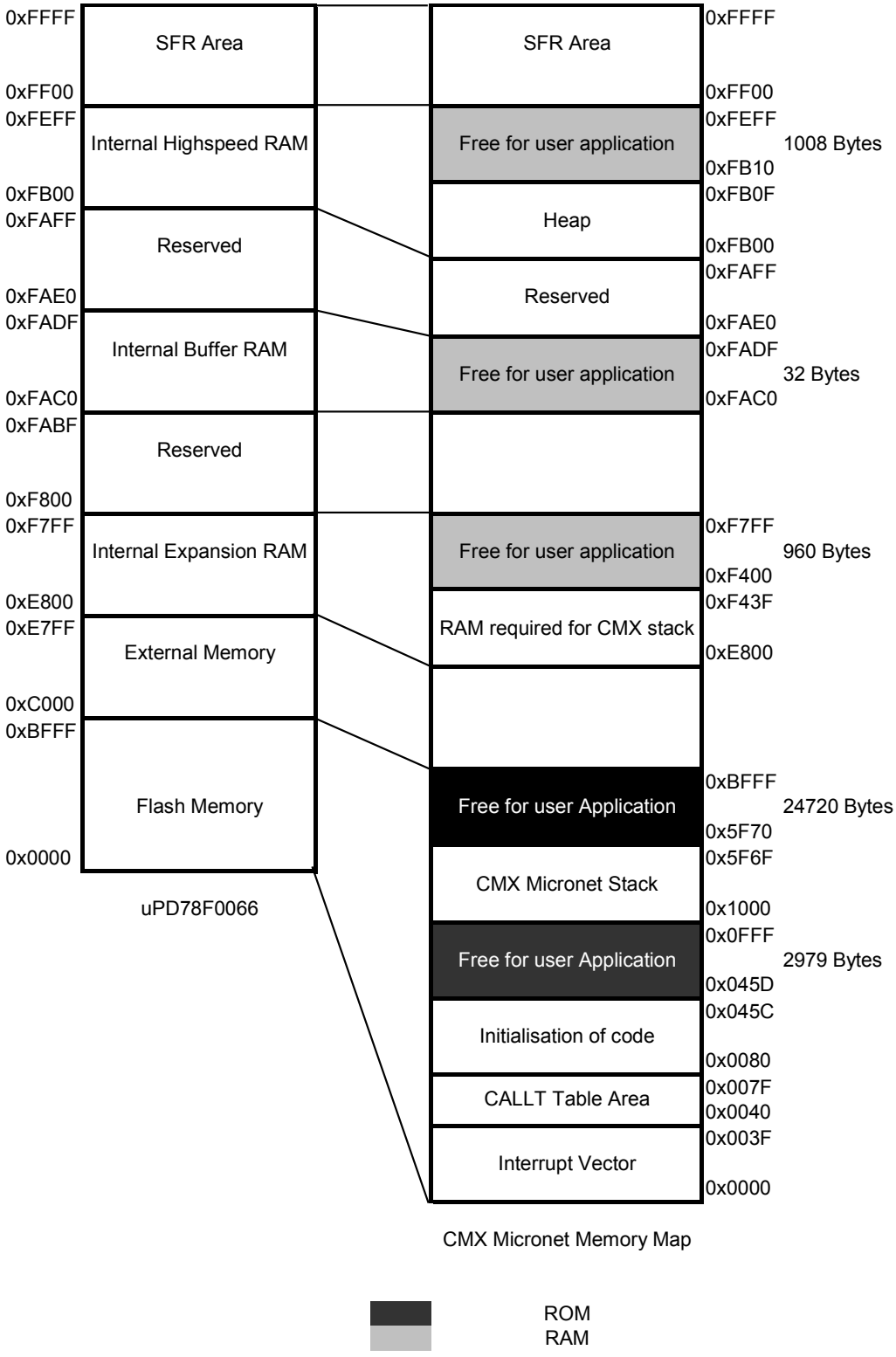
Source Code:

Example.c	-	Contains initialisation and startup routines for CMX-Micronet
Callback.c	-	IP address and additional function calls for CMX-Micronet
NEC_hw.c	-	All CMX-Micronet peripheral requirements
Init.c	-	Initialisation of hardware and software for user application

3.1. Memory Map

The 78K-Test It! CMX-Evaluation version memory layout is shown in the figure below.

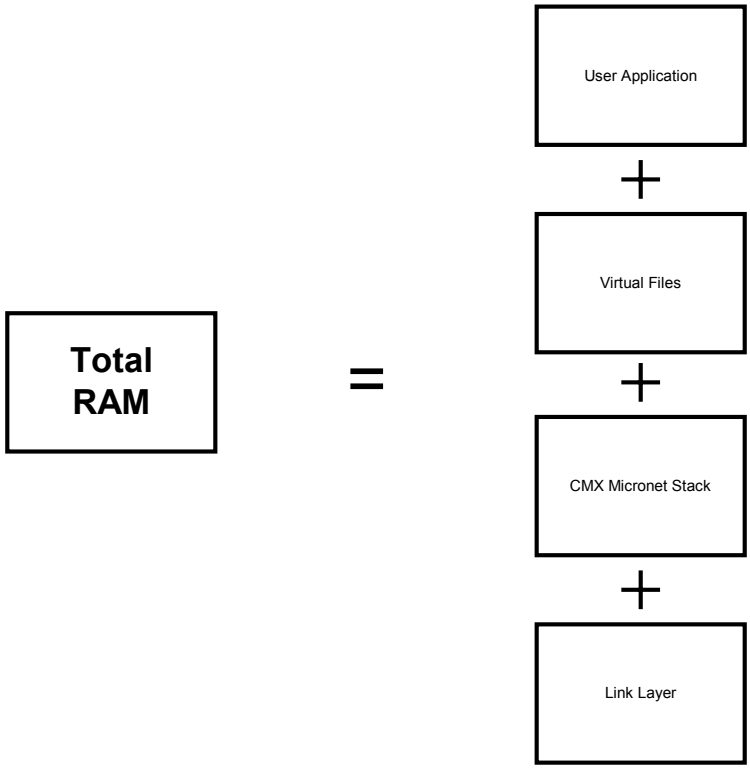
Figure 3-2 Memory Map



3.2. RAM

While the CMX-Micronet protocol stack has been designed for small RAM and ROM footprints, this does not mean that the requirement for RAM in a protocol stack goes away. The basis behind the Internet protocol stack has still come from the PC world where developers of software had plenty of RAM to use. Therefore Figure 3-3 shows that the total RAM requirement for the system is made up of 4 parts.

Figure 3-3 RAM Usage



3.2.1. Link Layer

The Physical Layer for PPP requires full duplex operation over a UART. The buffer size for both receive and transmit have a direct relationship to the size of a PPP packet and therefore the size of these buffers must be chosen to reflect this. The overall size of these buffers are dependant on the performance required for the user program and a compromise on the RAM available. To calculate a RAM value for the receive and transmit buffers use the following:

$$\text{RECV\_BUFF\_SIZE, XMIT\_BUFF\_SIZE} = 180 + x \qquad \text{(bytes)}$$

where: $0 \leq x \geq MAX$

MAX is determined by user program RAM requirements and performance required.

Remark: If smaller buffers are required it is possible to implement SLIP instead of PPP, but for evaluation purposes this has been disabled.

3.2.2. CMX Micronet Stack

The CMX Micronet Stack requires RAM for buffers within the network layer. It also requires RAM for Sockets and Virtual Files, which are stored as arrays at compile time. Therefore the actual value of RAM required for the protocol stack can vary greatly and a cautious designer will need to take this into account when implementing the end system. Therefore to understand this use the following:

$$CMX\_Micronet\_Stack = TCP\_WINDOW + HTTPTotal + ArrayTotal$$

where:	TCP_WINDOW	Buffers in TCP Layer
	HTTPTotal	Buffers in HTTP Application
	ArrayTotal	Compiled arrays in different layers

The following equation shows the relationship between the link layer receive and transmit buffers and the size of the network layer TCP buffers.

$$TCP\_WINDOW < RECV\_BUFF\_SIZE \quad \text{(bytes)}$$

$$\frac{RECV\_BUFF\_SIZE}{TCP\_WINDOW} = wholenumber \quad \text{ie: 2,3,…}$$

Remark: The equation shows that the TCP buffer must be smaller than the receive and transmit buffers to ensure that a complete TCP packet can be encapsulated within the link layer buffers. For successful operation the relationship between the TCP buffer and the link layer buffers must be a multiple; ie if RECV_BUFF_SIZE is 512 bytes then TCP_WINDOW can be 64,128, or 256 bytes.

HTTP Application requires 3 different buffers depending on the type of implementation required. The following equation shows how to calculate the total RAM requirement for this application:

$$HTTPTotal = URI\_BUFFER\_LEN + BODY\_BUFFER\_LEN + HTTP\_BUFFER\_LEN$$

where: URI_BUFFER_LEN is used to store URI data from Server Side Include requests. Default 64 bytes
BODY_BUFFER_LEN is used to store body data from POST requests. Default 64 bytes
HTTP_BUFFER_LEN size is set from the Virtual Files Section 3.2.3

Remark: Both of these buffers will need to be increased if more than 64 bytes are required for either type of function.

To help understand the RAM requirements for the arrays TABLE 3-1 shows the values of RAM required for each of the elements that make up the protocol stack.

Table 3-1 Array RAM usage

Layer	Function	Name	Bytes per number	Number
TCP	Sockets	NUM_SOCKETS	2 + 52 x Number	1 to 16
PPP	Password Authentication	PAP_NUM_USERS	Name + Password	1 to 255
FTP	Password Authentication	FTP_NUM_USERS	FtpName + FtpPassword	1 to 127
User Application	Virtual File Pages	NUM_VF_PAGES	6 + Name Length	0 to 255
User Application	Post Functions	NUM_POST_FUNCS	2 + Function Length	0 to 255
User Application	Server Side Includes	NUM_GET_FUNCS	2 + Function Length	0 to 255

where:

Name(default) = 10*
Password(default) = 10*
FtpName(default) = 10*
FtpPassword(Default) = 10*
Name Length(default) = 20*
Function Length(default) = 20*

Note: These figures denote the number of characters in a string

To determine the total RAM usage for the arrays use the following equation using the table above:

$$ArrayTotal = Socket\_RAM + PAP\_RAM + FTP\_RAM + FILE\_RAM + POST\_RAM + SSI\_RAM$$

where:

$$Socket\_RAM = NUM\_SOCKETS \times Number$$
$$PAP\_RAM = PAP\_NUM\_USERS \times Number$$
$$FTP\_RAM = FTP\_NUM\_USERS \times Number$$
$$FILE\_RAM = NUM\_VF\_PAGES \times Number$$
$$POST\_RAM = NUM\_POST\_FUNCS \times Number$$
$$SSI\_RAM = NUM\_GET\_FUNCS \times Number$$

3.2.3. Virtual Files

HTTP_BUFFER_LEN is one of the largest buffers required within CMX-Micronet. When Server Side Includes are enabled this buffer MUST be the same size as the largest HTML virtual file page*. If SSI is not enabled this buffer is optional to allow a header to be added to a HTML page. The default value would then be 64 bytes. For the evaluation version SSI's have been enabled.

Note: The size of the buffers is dependant on the HTML page text size, not the array size after conversion using html2c.exe.

3.2.4. User Program available RAM

In addition to the protocol stack RAM requirements the user program will require RAM and so it is important to understand how much RAM will be available for this task. For the evaluation system the following paramaters have been fixed to allow a certain amount of user program RAM. Use this example as guide to implement your own application.
Fixed Values for the Object Code Time limited version:

Sockets	6
Server Side Includes	8
POST Functions	8
Virtual File Pages	7
HTTP_BUFFER_LEN	1536
RECV_BUFF_SIZE	256
XMIT_BUFF_SIZE	256
URI_BUFFER_LEN	64
BODY_BUFFER_LEN	64
TCP_WINDOW	128

Check the following:

Link Layer

$$\text{RECV\_BUFF\_SIZE, XMIT\_BUFF\_SIZE} = 180 + x = 512$$

$$x = 332$$ (OK)

CMX Micronet Stack

$$\frac{RECV\_BUFF\_SIZE}{TCP\_WINDOW} = wholenumber = \frac{512}{128}$$

$$\frac{RECV\_BUFF\_SIZE}{TCP\_WINDOW} = 4$$ (OK)

$$HTTPTotal = 64 + 64 + 1536$$

$$HTTPTotal = 1664$$ (OK)

$$ArrayTotal = Socket\_RAM + PAP\_RAM + FTP\_RAM + FILE\_RAM + POST\_RAM + SSI\_RAM$$

$$= (6 \times 52) + 2 + 0 + 0 + 7 \times (6 + 20) + 8 \times (2 + 20) + 8 \times (2 + 20)$$
$$= 848$$

$$CMX\_Micronet\_Stack = TCP\_WINDOW + HTTPTotal + ArrayTotal + Temp\_Variables$$
$$CMX\_Micronet\_Stack = 128 + 1664 + 848 + 82$$
$$CMX\_Micronet\_Stack = 2722$$

Note: Temp variables are included from 2 example arrays still contained within Example.c

Therefore:

RAM available for user program is:

$$TotalRAM = User\_Program + CMX\_Micronet\_Stack + Link\_Layer$$

$$5Kbytes = User\_Program + 2722 + (256+256)$$

$$User\_Program = 5152 - 2722 - (256 - 256)$$
$$User\_Program = 1918$$

There is 1918 bytes available for User Program. This value includes stack requirements which has a default value of 848 bytes.

The stack value has been set to 848 bytes to accommodate CMX-Micronet on the 78K-Test It! Board. A high baud rate has been set and a low main clock frequency of 4.194304MHz is used. The stack value is modified at link time within the *.xcl file.

4 Hardware and Peripherals

This section describes the hardware requirements for the evaluation of CMX-Micronet. The chosen I/O, timers and interrupts are common to most 78K0 devices. Making the CMX-Micronet very portable across the 78K0 family. However the I/O utilises some of the hardware on the 78K-Test It! evaluation board.

4.1. µPD78F0066 Peripheral and I/O Assignments

Table 4-1 I/O Assignment

CN1	Port	RS232
1	--	NC
2	RXD0	RxD
3	TXD0	TxD
4	--	NC
5	--	Ground
6	--	NC
7	P77	CTS
8	P76	RTS
9	--	NC

Note: P70 output signal may be used to shut down the MAX3222 serial line driver. It is recommended to shut down the MAX3222 driver if the serial interface is not used. This will extend the battery lifetime. A low signal output at P70 will shut down the MAX3222. A high signal will wake up MAX3222. P71 output signal must be set to low signal to enable MAX3222.

4.2. Timer Functions

Timer 5 (TM50) 10ms Interval Timer for CMX-Micronet Stack operation
(This is fixed for the evaluation version, but can be changed within the source code version)

Note: This timer can be changed to suite other 78K0 devices. It can also be used for user application if a 10ms Interval timer is required. The location of this timer setup is contained in nec_hw.c.

4.3. Interrupts

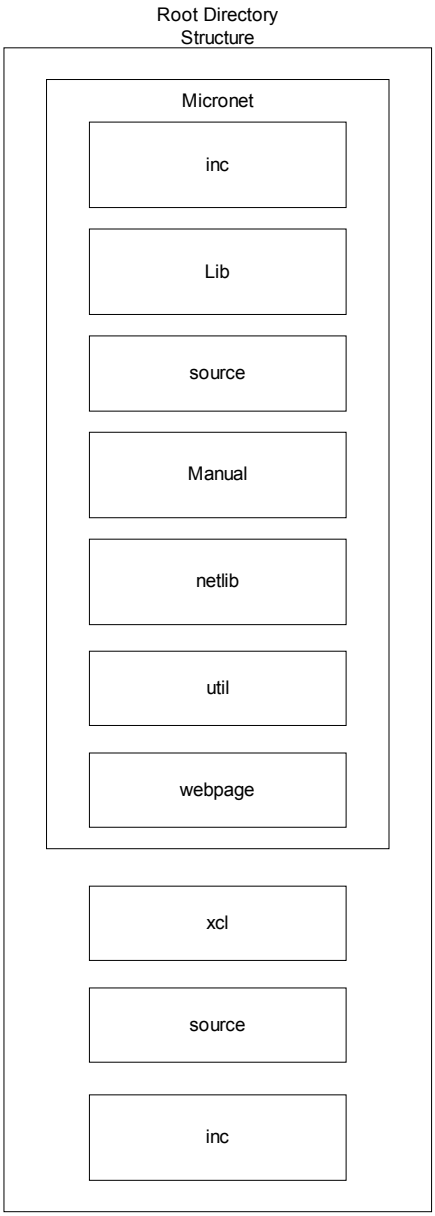
UART Recieve	(INTSR0)	Set for 38400 Baud, 8 bit, 1 stop, no parity @ 4.194304MHz
UART Transmit	(INTST0)	Set for 38400 Baud, 8 bit, 1 stop, no parity @ 4.194304MHz
UART Error	(INTSER0)	Enabled to check for framing and parity error

Note: These values can be changed within the nec_hw.c.

5 Description of Software

5.1. Software Location Setup

Figure 5-1 Directory Structure

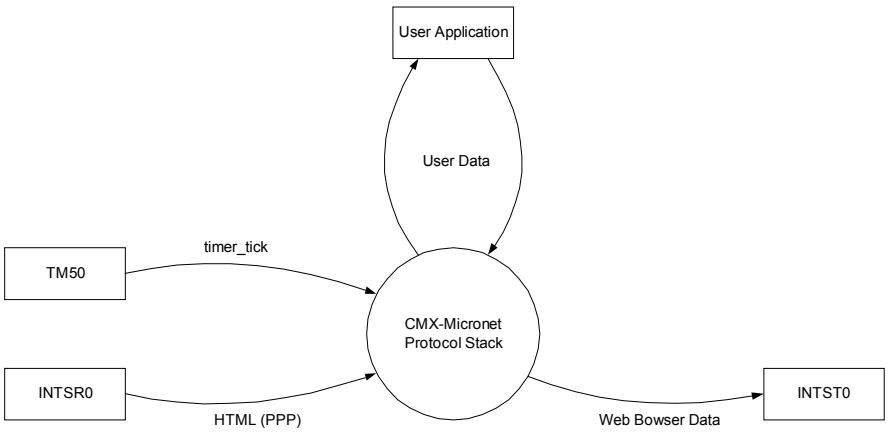


/Micronet/Inc	-	Location of all *.h files.
/Micronet/Lib	-	Location of object files from Micronet after library build. Micronet.r26
/Micronet/netlib	-	Location of all *.c files supplied by CMX if source build supplied
/Micronet/Manual	-	Location of user manual from CMX supplied with source build
/Micronet/Util	-	Location of Micronet utilities.
/Micronet/Webpage	-	Location of Webpage development.
/Micronet/source	-	Location of CMX source code for API build. Example.c,Callback.c
/source	-	Location of *.c files for user application including nec_hw.c
/xcl	-	Location of memory map requirements for build
/inc	-	Location of NEC *.h files used for application example

5.2. Context Diagram

The relationship between the software and hardware peripherals are shown in the context diagram Figure 5-2.

Figure 5-2 Context Diagram



5.3. Interaction with CMX-Micronet

There is no set location for a user application to interact with CMX-Micronet. There can only be pointers to use the protocol stack. These pointers are contained in Example.c and Callback.c. However there can be guides to where to put a user application depending on the type of interraction required.

For example a user application is required to do something and return a value depending on a submit button contained within a HTML page. This type of interaction is contained in the example of Electronic Dice shown in 7-2-1. The original example has come from the original 78K – Test It! program examples, where the example required an input via a key press to display a random value between 1 and 6 on the LED's provided.

To allow user interraction to occur using a Web browser the key press on the 78K-Test It! is substituted with a Web browser submit button press. So before a user application can interract with CMX-Micronet it is important to detirmine the type of interraction required. This task can be broken down into 3 different types of interaction.

Figure 5-3 User Application Active Only When Needed

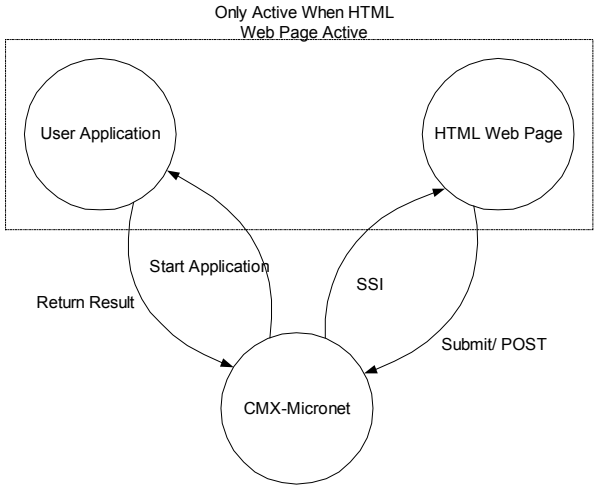


Figure 5-3 shows the type of connection required for the electronic dice example. The figure shows that the user application does not need to be active all the time, but only needs to be active when an action occurs from the HTML Web page.

In this case the interraction takes the form of a submit button within the HTML page. The PC client requests an update of the HTML page from the server using a submit button. The HTTP server then acknowledges the update and included within the HTML Web page are Server Side Include functions (#exec cgi=). The HTTP server acts on these SSI functons. The user application is called from the SSI function to get the next dice value(returned result). The returned result is formated within the SSI function call and a text substitution occurs within the HTML page.

The SSI functions used for this type of interraction are contained within Example.c.

Figure 5-4 User Application Active All The Time

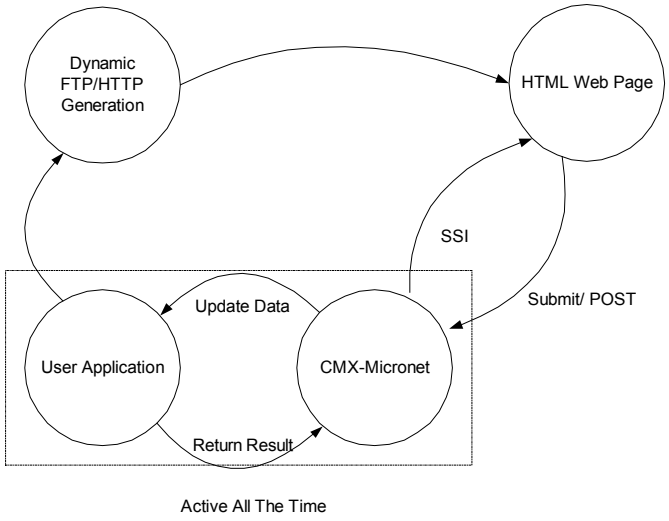


Figure 5-4 shows the user application to be running continuously. For this type of user application it is nessesary to use different options to run the user application in parallel to CMX-Micronet, depending on the type of application. These are:

1. An additional timer to operate the user application in parallel to the CMX-Micronet.
2. The user application call can be contained within the round robin used for checking for a PC connection not started shown in 5-4-2, or within the callback routines discussed in a later chapter.

The first option allows the user application to run in parallel to CMX-Micronet, even after a connection is established with a client. The second option only allows the user application to operate when the client has not established a connection.

The interraction to CMX-Micronet through the Callback.c and Example.c routines has its advantages for applications such as remote diagnostics where the application program must run all the time, and only data for status, logged information and control are needed.

It can also be used to dynamically develop HTML pages, or files for FTP download that can be stored in RAM. A future example will be developed using CMX-Micronet to output data to a Web browser based on the 78K-Test It! datalogger example.

Figure 5-5 User Application Using Both Types Of Interaction

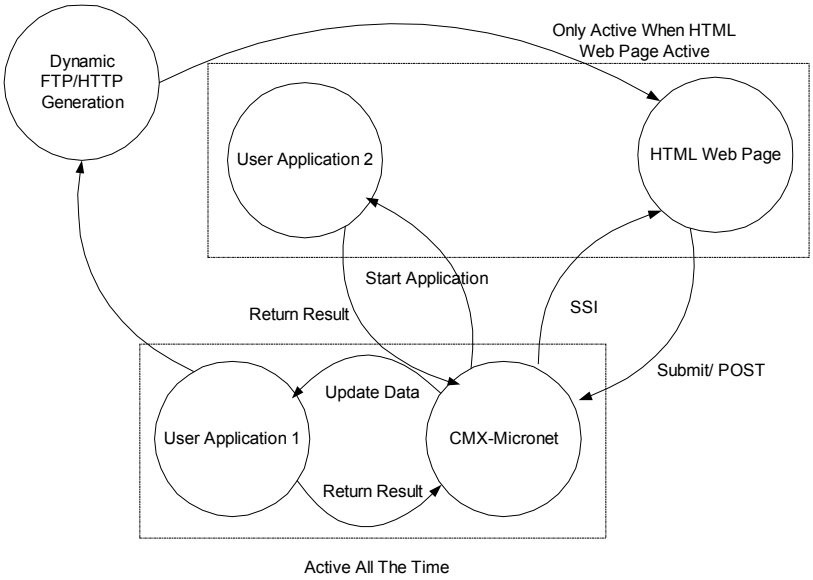


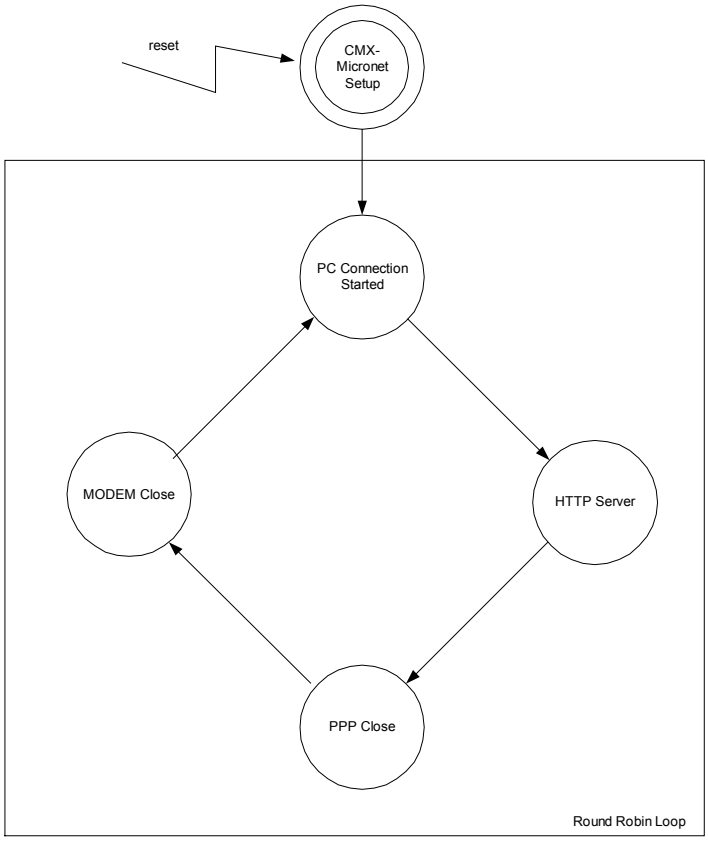
Figure 5-5 shows that both types of user application are possible within the one system.

The type of user application required for interraction to CMX-Mironet is important and should be the initial point of discussion before integration occurs, and the RAM requirements have been assigned.

5.4. State Diagram for Example.c

Figure 5-6 removes the peripherals from CMX-Micronet and shows the different processes involved in running an application layer such as HTTP server in what is simply termed a round robin approach for connection/disconnection to an external MODEM. To simplify the design the HTTP server can be replaced in Figure 5-6 with one of the following: TCP, UDP, FTP Server, or SMTP. For evaluation this has been limited to HTTP, or TCP if required.

Figure 5-6 State Diagram for Example.c



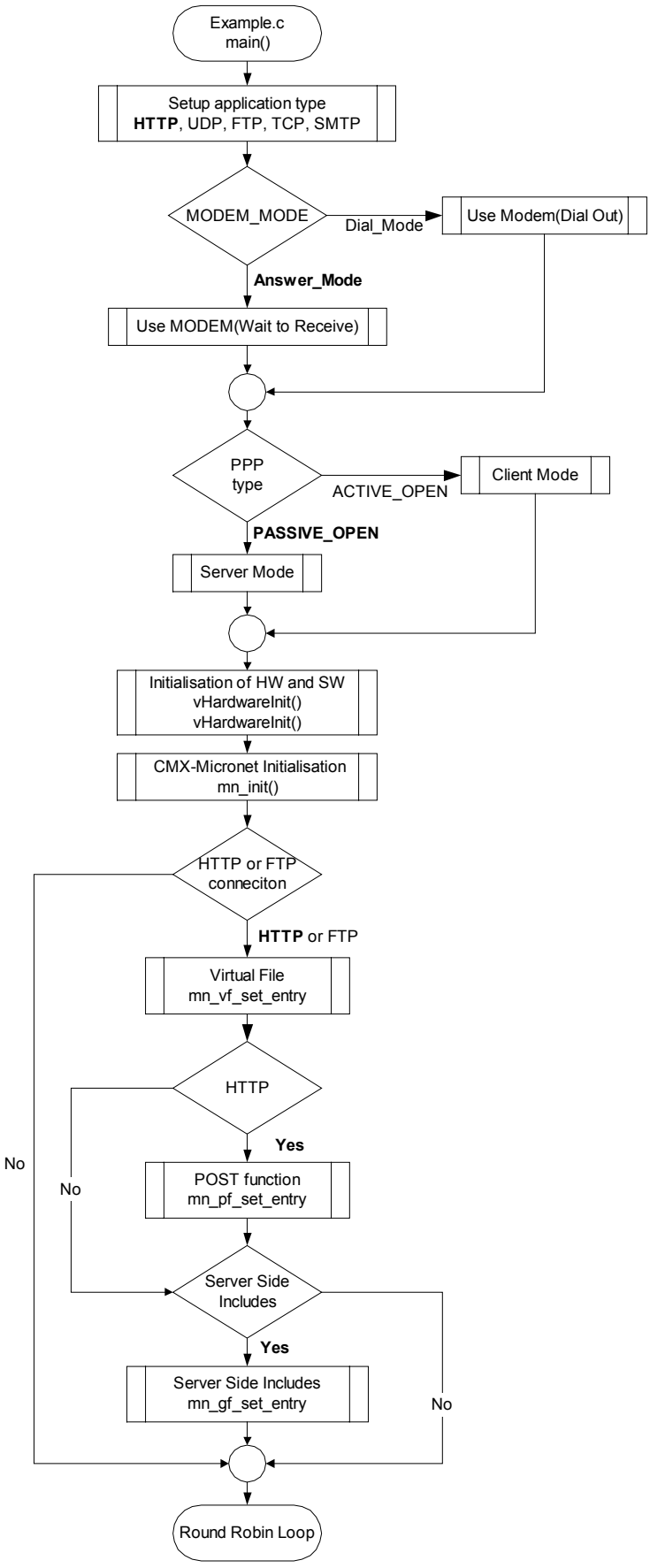
5.5. Software Discriptions

This sub section describes the various processes in the state machine shown in Figure 5-6.

5.5.1. CMX Micronet Setup

This process is responsible for initialising the μPD78F0066 after system reset and is the main entry point into CMX-Micronet and user application. It configures the type of connection that the CMX-Micronet will use and initialises the hardware and software for the task. Figure 5-7 shows the flow through initialisation, and the options in **bold** highlight the compile options used for the evaluation version of CMX-Micronet.

Figure 5-7 CMX-Micronet Setup Flow Diagram



The entry point(main()) after reset for the evaluation software is contained in example.c, with each of the configuration options for CMX-Micronet pre-set within config.h.

The header config.h contains all configurable options for setting the protocols used, number of sockets, sizes of buffers, etc. These configuration options can be changed within the full source code version. For the evaluation version these values have been fixed. See 3.2.4 for the fixed values.

5.5.1.1. Switch Settings

Additional configuration options have been included within example.c:

Where:

- Application type(EXAMPLE_TYPE) is set to use HTTP,
- MODEM connect type(MODEM_MODE) is set to be Answer_Mode. Where in Answer Mode the device running CMX-Micronet is waiting for a call from a MODEM. It is possible within the full source code version to dial out. MODEM_MODE would be set to Dial_Mode for this option.
- PPP Type(open_mode) is set to PASSIVE_OPEN for server mode. This means that the client must first initiate the connection to the CMX-Micronet device. To initiate a connection from the CMX-Micronet device this option can be set to ACTIVE_OPEN and the CMX-Micronet device becomes a client.

5.5.1.2. User Application Initialisation

Initialisation of Hardware and Software is contained in the functions vHardwareInit() and vSoftwareInit() respectively. Both of these functions are to initially set up the device peripherals and clocking for user application only. Both functions are located in init.c.

5.5.1.3. CMX-Micronet Initialisation

Before any API function calls are made to CMX-Micxronet the protocol stack must be initialised by calling mn_init().

5.5.1.4. Virtual File Setup

After initialisation of CMX-Micronet, hardware and software for the user application it is nessesary to initialise any Virtual File pages required for either FTP, or HTTP use. This is done with the function call mn_vf_set_entry() where each file, or page required for entry must use a separate function call. For the evaluation version only 7 files are allowed. The format for the Virtual File Pages is outlined in 6-8.

Note: For every Virtual File page whether it is a graphic, or HTML page will need to be pass through the conversion program html2c.exe. This formats the data into an array with an attached header that can be compiled into the device containing CMX-Micronet at compile time. The headers have to be included at the top of example.c.

5.5.1.5. POST Function Setup

Post Function Setup is required for those Web pages developed with POST functions enabled. These function calls allow user interaction between the HTML Web pages and the user application. The function call for POST function is mn_pf_set_entry(). A separate function call is required for each POST function used within the HTML Web pages. For the evaluation version only 8 POST functions are available.

Note: Once a POST function has been initialised there must be a corresponding function developed. For the examples provided this is also contained in Example.c.

5.5.1.6. Server Side Include Setup

SSI setup is required when a user program requires interaction with the HTML pages. This allows a direct substitution of text within a HTML Web page. The function call for this is mn_gf_set_entry() . Only 8 SSI functions are available for the evaluation version.

Note: Like the POST function the SSI function initialisation also requires a corresponding function to be developed. In both cases each of these functions are specific to the user application requirements, or action needed for the task required.

5.5.2. PC Connection Started

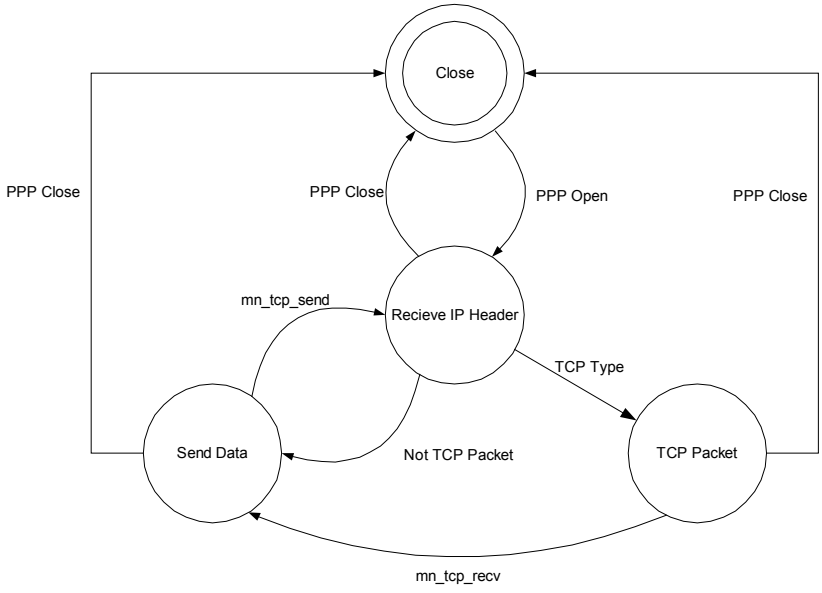
In this evaluation version the MODEM function has been set for a direct connection to a PC. This means that mn_modem_connect() type has been set to DIRECT_CONNECT and the NULL_MODEM option is set to one. This allows connection to a PC through a null MODEM cable and a null MODEM driver must be used within the PC. This option is supported on Windows NT, or Windows 2000/ME/XP. See Getting Started User Manual for instructions on the connection to a PC.

Based on the initial settings for the MODEM connect type this is the entry point to the round robin loop shown in Figure 5-3. A test is carried out within the mn_modem_connect() function, if a MODEM connection is successful entry is allowed into the round robin. If the test is not successful the round robin is not entered and within example.c a loop is generated to keep testing for a MODEM connection using the mn_modem_connect() function.

5.5.3. HTTP Server

Figure 5-8 decomposes the HTTP Server process to highlight that once a successul PPP negociation has taken place CMX-Micronet will continually loop within this application layer. Once the client requests a PPP close the HTTP Server will exit and wait for the client to re-establish a PC connection as shown in Figure 5-6.

Figure 5-8 Simplified State Diagram for HTTP Server



5.5.4. PPP Close

After HTTP Server has shut down due to the client requesting a disconnect the link layer must be shut down from the server side using the function call mn_ppp_close().

5.5.5. MODEM Close

The MODEM must also be shut down from the server side using the function call mn_modem_disconnect(). Once the MODEM has been shut down this completes the round robin loop.

6 Application Programmers Interface (API)

This chapter is not a definitive list of all function calls and procedures available within CMX-Micronet. It is only intended to expand on the procedures and function calls used in the previous chapters for evaluation. If more detailed information is required to customise how the CMX-Micronet stack is used then the full CMX-Micronet User Manual would be provided with the source code version.

6.1. Initialisation

mn_init()

This function initialises CMX-Micronet. This function must be called before any CMX-Micronet function.

6.2. HTTP

mn_http_server()

This function opens a TCP socket and waits for a HTTP request from the client. Additional sockets are opened and closed as and when required to service client requests. This function does not close down unless there is an error in opening a socket, or PPP connection is shut down by the client as shown in Figure 5-5. The callback function mn_app_server_idle() when no packets are being processed.

mn_http_init_recv()

This function performs initialisation if HTTP variables when receiving data starts. It is also called from the Callback function mn_app_init_recv().

mn_http_recv_byte(c)

Where c is the latest byte retreived from the HTTP packet.

This function processes a received byte. It is called from the Callback function mn_app_recv_byte(). The type of HTTP request is detirmined. There are three types of acceptable requests: SSI(GET), HEAD and POST.

mn_http_process_packet(PSOCKET_INFO)

Where PSOCKET_INFO is the socket pointer

This function processes a received packet. It is called from the Callback function mn_app_process_packet(). If the HTTP is SSI(GET), or HEAD the function will use the mn_vf_get_entry() function to obtain a HTML Web page located in the URIbuffer. If the request is a POST function the mn_pf_get_entry() is used to lookup and run the function from the URIbuffer.

mn_http_find_value(BODYptr, name, buff)

Where BODYptr points to the POST body whwere we are looking for a match
Name is the field we are looking for
Buff is the buffer where the found value is being stored

Returned Status
1: The function call was successful
0: Field name not found

This function searched the field name = field value pairs for the passed field name and copies the decoded field value into the passed buffer. POST functions would use this function to detirmine the value of variables sent from the Web page.

mn_http_process_includes(PSOCKET_INFO)

Where PSOCKET_INFO is the socket pointer

If SSI's are enabed this function is used to inser values directly into a Web page. The page is searched for strings in the form: <!--#exec cgi="userfunc"--> The name inside the quotation marks is looked for in the get function table mn_gf_get_entry function and the associated function is run.

6.3. TCP

mn_tcp_open(type, PSOCKET_INFO)

Where type is either ACTIVE_OPEN for client mode, or PASSIVE_OPEN for server mode. PSOCKET_INFO is the socket pointer.

Returned Status

TCP_ESTABLISHED:	The function call was successful
TCP_CLOSED:	Unable to establish a connection
SOCKET_NOT_FOUND	Invalid socket pointer passed to function

This function is used to establish a TCP connection. All initialisation must be done and MODEM and PPP connection must be established before calling this function.

mn_tcp_send(PSOCKET_INFO)

Where PSOCKET_INFO is the socket pointer

Returned Status

Number of bytes sent:	Function call was successful
0:	No data to send
TCP_ERROR:	Unable to send packet
TCP_TOO_LONG:	Packet is too long tp send in one piece

This function sends a packet using data pointed to by the send_ptr.

mn_tcp_recv(PSOCKET_INFO *)

Where PSOCKET_INFO is a pointer to the socket pointer

Returned Status

Number of bytes received:	Function call was successful
NEED_TO_LISTEN:	A reply to the received packet was atomatically sent and should wait for an answer.
TCP_BAD_HEADER:	The received packet had a bad header
TCP_BAD_CSUM:	The TCP checksum failed
TCP_BAD_FCS:	The PPP frame check sequence failed.

This function receives a TCP packet from a remote TCP. The received packet will be assigned to the correct socket, a new socket is opened if necessary.

mn_tcp_shutdown(PSOCKET_INFO)

Where PSOCKET_INFO is the socket pointer

This function closes the current TCP connection for the passed socket. This function should be used when the connection with the remote is working correctly. If the other side is not responding the function mn_tcp_abort should be used instead.

mn_tcp_abort(PSOCKET_INFO)

Where PSOCKET_INFO is the socket pointer

This function reset the TCP statemachine for the current passed socket. This function should not be used if the remote is working correctly.

6.4. IP

mn_ip_recv()

Returned Status

0:	No packet received or the packet received had an IP address that was not one of ours.
Greater than zero:	one of the packet types defined in ip.h was found.

This function receives either an Ethernet, PPP, or SLIP header and the IP header. IF an ICMP request is received it will send an ICMP reply if a PING request is sent.

6.5. PPP

mn_ppp_close()

This function closes a PPP connection without waiting for a response and resets the PPP state machine.

mn_ppp_open(type)

Where Type is either ACTIVE_OPEN, or PASSIVE_OPEN depending on client or server mode.

Returned Status

1:	Function call wa successful
0:	Could not establish a connection

This function is used ot establish a PPP connectionwith a remote PPP. Before this can take place all initialisation must be done including the MODEM connection.

mn_ppp_reset()

This function resets the PPP statemachine. This function should only be called if an error condition exists.

6.6. Callback

mn_app_get_send_size(PSOCKET_INFO)

Where PSOCKET_INFO is the socket pointer

Returned Status

Number of bytes to send in current packet.

This function is called from various other functions to get the number of bytes of data to be sent.

mn_app_http_server_idle()

Returned Status

NEED_TO_SEND: If data must be sent immediately by the server.

This function is used for application speific program when the Web server is not processing any packets.

mn_app_init_recv(ip_recv_len, PSOCKET_INFO)

Where ip_recv_len is the length of the incomming packet not including headers
PSOCKET_INFO is the current socket pointer.

This function is used to allow any specific initialisation to occur before receiving data from a packet.

mn_app_process_packet(PSOCKET_INFO)

Where PSOCKET_INFO is the current socket pointer.

This function ia used to allow processing of a packet after it is done being received.

mn_app_recv_byte(c, PSOCKET_INFO)

Where c is the character just received from the incoming TCP, or UDP packet
PSOCKET_INFO is the current socket pointer.

This function is called to allow processing of each single byte of data as it is received.

mn_app_recv_idle()

Returned Status
NEED_TO_EXIT: If mn_recv must exit now

This functionis called when mn_recv is not processing any packets. Any user application code can be placed within this function when mn_recv is idle.

6.7. MODEM

mn_modem_connect(byte)

Where byte: Is either ANSWER_MODE if MODEM is waiting for a call, or DIAL_MODE the MODEM is dialing out.

Returned Status

1:	Function call was successful
negative number:	Could not establish a connection

This function connection the CMX-Micronet device to another MODEM.

mn_modem_disconnect()

This function puts the MODEM into command mode and hangs up the MODEM.

mn_modem_send_string(const byte *, word16)

Where	const byte *:	Is the pointer to the MODEM string
	word116:	Length of the reply

This function sends a string of a passed length to the MODEM.

mn_modem_wait_reply(const byte *, word16, word16)

Where	const byte *:	Is the string to get from the MODEM
	word16:	Is the length of the string to get
	word16:	is the number of system ticks to wait for string.

Returned Status

1:	function call was successful
-1:	could not establish a connection

6.8. Virtual File System

mn_vf_get_entry(byte *)

Where byte *: is the name of the Web page to look up

Returned Status

A pointer to the directory table entry corresponding to the passed name.

This function returns a pointer to the directory entry crresponding to the passed file name.

mn_vf_set_entry(„mypage.htm“, page_size, page_ptr, page_type)

Where „mypage.htm“ is the name of the page to insert into the virtual file table
Page_size is the number of bytes in the page pointed to by page_ptr
page_ptr is the address of the Web page to be added to the table
page_type is either VF_PTYPE_DYNAMIC for the file to be stored in RAM, VF_PTYPE_STATIC for the file to be stored in ROM, and VF_PTYPE_FLASH for the file to be stored in programable flash.

Returned Status

A pointer to the directory table entry corresponding to the passed name.

This function places the passed file name, size, location and page type into the first empty directory if one is available.

mn_vf_del_entry(name)

Where name is the page to delete

Returned Status

1: page deleted successfully
0: page not found

This function is used to delete, or remove a directory entry.

6.8.1. HTML

It must be possible to add Web pages to your embedded application. To do this you first must develop the Web pages using a HTML editor. The HTML pages must be tested on a browser such as Explorer, or Netscape before integration into the embedded application.

To view the pages developed within the example activate the following hyperlink:
..\MICRONET_V2_15f\Webpage\index.htm

ALL files generated by the HTML editor must be included within the embedded application.

Here is a copy of the source code generated by the developed HTML Web page example used for the electronic dice.

HTML Source generated by Explorer

```
<HTML><HEAD><TITLE>CMX MicroNet Demo</TITLE>
</HEAD><BODY bgcolor="#FFFFFF" onLoad="window.status='78K-Hopper Demo'">
<CENTER>
  <p><font face="Georgia, Times New Roman, Times, serif"><b><i><font size="7" color="RED">78K-Hopper
    (EWS78K-TESTIT)</font></i></b></font></p>
  <p><b><i><font face="Georgia, Times New Roman, Times, serif" size="6" color="BLUE">This
    page is coming from 78K-Hopper.</font></i></b></p>
```

```
  <p>
<HR></center>
<p><font size="4" color="BLUE">78K-Hopper is running CMX-Micronet from CMX systems.
  It can display more than just HTML pages. Forms can be added to send data can
  be acted apon by user defined functions.</font></p>
<center>
  <font color="#0000FF">Here is a simple demonstration for an Electronic Dice</font>
  <FORM NAME="LCD_TEXT" ACTION="lcd_msg" METHOD=POST>

    <p> <font size="7">
      <!--#exec cgi="dice"-->
      </font> </p>
    <INPUT TYPE="Submit" VALUE="Roll Dice" name="display"">
  </FORM>
  <FORM NAME="LCD_TEXT" ACTION="lcd_stop" METHOD=POST>

    <input type="Submit" value="Reset" name="Stop"">
    </FORM><HR>
</center>
<div align="center">
  <p> </p>
  <p><img src="table.gif" width="788" height="66"></p>
</div>
<p><img src="necem.gif" width="400" height="60"></p>
<p align="right"><FONT SIZE="4"><a href="page2.htm">Go to Page 2</a></font><br>
</p>
<CENTER>
</CENTER>
</BODY></HTML>
```

To get HTML generated files into a format required for integration ALL files must pass through the html2C.exe application contained in the *util* directory. ALL files must be placed into a flattened directory structure before integration into the user application.

Below shows how to integrate developed Web pages into your embedded application using the mn_vf_set_entry() function to create the virtual file structure.

Sample HTML code integration from Example.c

```
/* add Web pages to virtual file system. The main page MUST be first */
    mn_vf_set_entry((byte *)"index.htm", INDEX_SIZE, (void *)index_htm,VF_PTYPE_STATIC);
    mn_vf_set_entry((byte *)"page2.htm", PAGE2_SIZE, (void *)page2_htm,VF_PTYPE_STATIC);
    mn_vf_set_entry((byte *)"necem.gif", NECEM_SIZE, (void *)necem_gif,VF_PTYPE_STATIC);
    mn_vf_set_entry((byte *)"table.gif", TABLE_SIZE, (void *)table_gif,VF_PTYPE_STATIC);
```

ALL the files above require NO interaction from the browser to the embedded application. The embedded application knows the location of the main page and sends this to the browser. ALL following pages are called by main page to be transferred to the browser.

6.8.1.1. Server Side Includes(SSI)

mn_gf_get_entry(name_ptr)

Where name_ptr has the name of the function to look uf

Returned Status
A function pointer corresponding to the passed name

This function returns a function pointer corresponding to the passed function name.

mn_gf_set_entry(„new_name“, new_func)

Where „new_name“ is the name of the function to be inserted into the table
new_func is the function to be added to the table.

Returned Result
A pointer to the SSI(GET) function table entry corresponding to the name passed.

This function places the passed name and function pointer into the first empty get function entry.

mn_gf_del_entry(func_name)

Where func_name has the name of the function to delete from the table

Returned Status

1: GET function entry successful
0: GET function entry not found

This function deletes the GET function entry corresponding to the passed function name.

6.8.1.2. POST Functions

mn_pf_get_entry(URIptr)

Where URIptr has the name of the function to look up.

Returned Status
A function pointer corresponding to the passed name.

mn_pf_set_entry(„new_name“, new_func)

Where „new_name“ is the name of the function to insert into the table
New_func is the function to be added to the table

Returned Status
A pointer to the POST function table entry corresponding to the passed name.

This function places the passed name and function pointer into the first POST function entry point.

mn_pf_del_entry(func_name)

Where func_name is the name of the function to be deleted from the table.

This function deletes the POST function entry.

7 Example Code

This section contains the souce code to interact with the API and source code for the application examples provided. The code is contained in various files based on the directory structure outlined in 5-1.

7.1. API Build

The API build for CMX-Micronet is contained below. Example.c is used for the initialisation of CMX-Micronet, Callback.c is used for the setup of IP addresses and for the additional routines. Init.c initialises the hardare and software required for the 78K0 device. NEC_hw.c contains the peripheral setup required for CMX-Micronet successful operation.

7.1.1. Example.c

```
/*********************************************************
Copyright (c) CMX Systems, Inc. 2001. All rights reserved
*********************************************************/

#include "micronet.h"
#include "User_config.h"

#define SERVER_MODE       1    /* set to 1 if a server, or 0 if a client */
#define MODEM_MODE      ANSWER_MODE    /* set to DIAL_MODE or ANSWER_MODE */

#define UDP_EXAMPLE       0
#define TCP_EXAMPLE       1
#define HTTP_EXAMPLE      2
#define FTP_EXAMPLE       3
#define SMTP_EXAMPLE      4

/*********************************************************
   To setup the example program for the UDP echo example
   make sure UDP is selected in config.h then change the
   line of code below to:
   #define EXAMPLE_TYPE    UDP_EXAMPLE

   To setup the example program for the TCP echo example
   make sure TCP is selected in config.h then change the
   line of code below to:
   #define EXAMPLE_TYPE    TCP_EXAMPLE

   To setup the example program for the HTTP server example
   make sure TCP and HTTP are selected in config.h then
   change the line of code below to:
   #define EXAMPLE_TYPE    HTTP_EXAMPLE

   To setup the example program for the FTP server example
   make sure TCP and FTP are selected in config.h then
   change the line of code below to:
   #define EXAMPLE_TYPE    FTP_EXAMPLE

   To setup the example program for the SMTP server example
   make sure TCP and SMTP are selected in config.h then
   change the line of code below to:
   #define EXAMPLE_TYPE    SMTP_EXAMPLE
*********************************************************/
#define EXAMPLE_TYPE    HTTP_EXAMPLE

#if !UDP && EXAMPLE_TYPE == UDP_EXAMPLE
#error UDP example selected without UDP
```

```
#endif
#if !TCP && EXAMPLE_TYPE == TCP_EXAMPLE
#error TCP example selected without TCP
#endif
#if !HTTP && EXAMPLE_TYPE == HTTP_EXAMPLE
#error HTTP example selected without HTTP
#endif
#if !FTP && EXAMPLE_TYPE == FTP_EXAMPLE
#error FTP example selected without FTP
#endif
#if !SMTP && EXAMPLE_TYPE == SMTP_EXAMPLE
#error SMTP example selected without SMTP
#endif


#if ((EXAMPLE_TYPE == HTTP_EXAMPLE) || (EXAMPLE_TYPE == FTP_EXAMPLE))
/* put #includes for Web pages here */
#endif    /* ((EXAMPLE_TYPE == HTTP_EXAMPLE) || (EXAMPLE_TYPE == FTP_EXAMPLE)) */


#if (EXAMPLE_TYPE == HTTP_EXAMPLE)
#undef SERVER_MODE
#define SERVER_MODE    1

void lcd_msg_func(PSOCKET_INFO) cmx_reentrant;
void lcd_stop_func(PSOCKET_INFO) cmx_reentrant;
#if SERVER_SIDE_INCLUDES
/* put #includes for SSI's here */
word16 getMsg_func(byte *) cmx_reentrant;
#endif


#elif (EXAMPLE_TYPE == FTP_EXAMPLE)
/* for now ftp is only server, this will change later */
#undef SERVER_MODE
#define SERVER_MODE    1

#elif (EXAMPLE_TYPE == SMTP_EXAMPLE)
/* smtp is only client */
#undef SERVER_MODE
#define SERVER_MODE    0
#else
static byte TESTSTRING[]= "THIS IS A TEST STRING TO ECHO";
#define DATA_BUFF_LEN    40
byte data_buff[DATA_BUFF_LEN];
#endif    /* (EXAMPLE_TYPE == HTTP_EXAMPLE) */

#if (defined(__BORLANDC__) || defined(_MSC_VER))
extern void (__interrupt __far *old1b)();        /* to hold old int 0x1b handler */
#endif

#if (EXAMPLE_TYPE == UDP_EXAMPLE)
void doUDP(void) cmx_reentrant;
#elif (EXAMPLE_TYPE == TCP_EXAMPLE)
void doTCP(void) cmx_reentrant;
#elif (EXAMPLE_TYPE == SMTP_EXAMPLE)
void doSMTP(void) cmx_reentrant;
#endif        /* (EXAMPLE_TYPE == UDP_EXAMPLE) */

#if BOOTP
   BOOTP_INFO_T bp;
#endif        /* BOOTP */
```

```
#if TFTP
#define TESTLEN    64
byte testBuffer[TESTLEN];
long length;
#endif        /* BOOTP */

static byte msg_buff[17];
const unsigned char Welcome[] =
"Welcome to NEC\r\n";


/* ---------------------------------------------------------------------- */
void main(void)
{
#if MODEM
   byte connect_mode;
#endif

#if PPP
   byte open_mode;
#endif

#if MODEM
#if (MODEM_MODE == ANSWER_MODE)
   connect_mode = ANSWER_MODE;
#else
   connect_mode = DIAL_MODE;
#endif
#endif    /* MODEM */

#if PPP
#if SERVER_MODE
   open_mode = PASSIVE_OPEN;
#else
   open_mode = ACTIVE_OPEN;
#endif
#endif        /* #if PPP */

#if (defined(POL78K0) || defined(CMX78K0))
   /*  put your hardware initialisation routines here for all peripherals and sfr's Except Timer and UART used
by stack*/
   vHardwareInit();
   vSoftwareInit();                  // Variable Initialization

#endif

   /* call mn_init before using any other MicroNet API functions */
   mn_init();

#if (defined(__BORLANDC__) || defined(_MSC_VER))
   old1b = getvector(0x1b);
   setvector(0x1b,onbreak);        /* stop ctrl-break */
   setvector(0x23,onbreak);
   atexit(cmx_restore);
#endif        /* #if (defined(__BORLANDC__) || defined(_MSC_VER)) */

#if ((EXAMPLE_TYPE == HTTP_EXAMPLE) || (EXAMPLE_TYPE == FTP_EXAMPLE))
   /* add Web pages to virtual file system. The main page MUST be first */


#if (EXAMPLE_TYPE == HTTP_EXAMPLE)
   /* add post functions to be used with forms */
   strcpy((char *)msg_buff,(char *)Welcome);
```

```
    mn_pf_set_entry((byte *)"lcd_msg", lcd_msg_func);
    mn_pf_set_entry((byte *)"lcd_stop", lcd_stop_func);

#if SERVER_SIDE_INCLUDES
    /* add any get functions (server-side-includes) here */
    mn_gf_set_entry((byte *)"getMsg", getMsg_func);

#endif        /* SERVER_SIDE_INCLUDES */
#endif        /* (EXAMPLE_TYPE == HTTP_EXAMPLE) */
#endif    /* ((EXAMPLE_TYPE == HTTP_EXAMPLE) || (EXAMPLE_TYPE == FTP_EXAMPLE)) */

while(1)
{
#if MODEM
    if (mn_modem_connect(connect_mode) < 0)
        EXIT(1);
#endif        /* MODEM */

#if PPP
#if USE_PAP
   mn_ppp_add_pap_user("cmx","password");
#endif      /* USE_PAP */

   if (!mn_ppp_open(open_mode))
        {
#if MODEM
        mn_modem_disconnect();
#endif    /* MODEM */
        EXIT(1);
        }
#endif        /* #if PPP */

#if DHCP
#define LEASE_TIME    1000
    if (mn_dhcp_start(PTR_NULL, LEASE_TIME) == 1)
        {
#if TFTP
        length = mn_tftp_get_file(dhcp_lease.server_id, dhcp_info.file, \
                testBuffer, TESTLEN);
            if (length < 0)        /* TFTP error */
                EXIT(1);
#endif        /* TFTP */
        }
    else            /* dhcp error */
        EXIT(1);
#elif BOOTP
    if ( mn_bootp(PTR_NULL,&bp) > 0)
        {
        /* if the #define BOOTP_REQUEST_IP was set to 0, should check here if
            bp.yiaddr is different than ip_src_addr.
        */
#if TFTP
        length = mn_tftp_get_file(bp.siaddr, bp.file, testBuffer, TESTLEN);
         if (length < 0)        /* TFTP error */
            EXIT(1);
#endif        /* TFTP */
        }
    else                        /* BOOTP error */
        EXIT(1);
#elif TFTP                /* tftp without bootp or dhcp */
    /* replace boot.mn with the file you wish to retrieve */
    length = mn_tftp_get_file(ip_dest_addr, (byte *)"boot.mn", testBuffer, TESTLEN);
     if (length < 0)        /* TFTP error */
```

```
      EXIT(1);
#endif       /* BOOTP */

#if (EXAMPLE_TYPE == UDP_EXAMPLE)
   doUDP();
#elif (EXAMPLE_TYPE == TCP_EXAMPLE)
   doTCP();
#elif (EXAMPLE_TYPE == HTTP_EXAMPLE)
   mn_http_server();       /* see http.c */
#elif (EXAMPLE_TYPE == FTP_EXAMPLE)
   mn_ftp_server();       /* see ftpservr.c */
#elif (EXAMPLE_TYPE == SMTP_EXAMPLE)
   doSMTP();
#endif       /* (EXAMPLE_TYPE == UDP_EXAMPLE) */

#if DHCP
   mn_dhcp_release();
#elif PPP
   mn_ppp_close();
#endif       /* #if PPP */

#if MODEM
   mn_modem_disconnect();
#endif    /* MODEM */

}
   EXIT(0);
}

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

#if (EXAMPLE_TYPE == UDP_EXAMPLE)
void doUDP(void)
cmx_reentrant {
   CHAR socket_no;
   PSOCKET_INFO socket_ptr;
   byte *data_ptr;
   word16 data_len;
   int status;

#if SERVER_MODE
   socket_no = mn_open(ip_dest_addr,ECHO_PORT,0,PASSIVE_OPEN,PROTO_UDP,\
                       STD_TYPE,data_buff,DATA_BUFF_LEN);
   data_ptr = PTR_NULL;
   data_len = 0;
#else       /* client mode */
   socket_no = mn_open(ip_dest_addr,DEFAULT_PORT,ECHO_PORT,ACTIVE_OPEN,PROTO_UDP,\
                       STD_TYPE,data_buff,DATA_BUFF_LEN);
   data_ptr = TESTSTRING;
   data_len = strlen((char *)TESTSTRING);
#endif

   if (socket_no < 0)    /* did we open socket successfully? */
      EXIT(1);

   socket_ptr = MK_SOCKET_PTR(socket_no);       /* get pointer to the socket */

   while(1)
      {
#if (defined(__BORLANDC__) || defined(_MSC_VER))
     if (_bios_keybrd(_KEYBRD_READY))     /* check if ctrl-c pressed */
         {
         if ((_bios_keybrd(_KEYBRD_READ) & 0xff) == 3)
```

```
            {
            mn_close(socket_no);
            break;
            }
          }
#endif       /* #if (defined(__BORLANDC__) || defined(_MSC_VER)) */

      status = 0;
      if (data_ptr != PTR_NULL)
         status = mn_send(socket_no,data_ptr,data_len);
      if (status < 0)
               {
            data_len = 0;
         break;
            }
      status = mn_recv(socket_no,data_buff,DATA_BUFF_LEN);
      if (status < 0 && status != SOCKET_TIMED_OUT)
               {
            data_len = 0;
         break;
            }
      /* if we got something, send back what we got */
      if (status > 0)
         {
         data_ptr = socket_ptr->recv_ptr;
         data_len = socket_ptr->recv_len;
         }

#if PPP
      if (!(ppp_status.up))
         break;
#endif    /* PPP */
      }

   mn_abort(socket_no);
}
#endif

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

#if (EXAMPLE_TYPE == TCP_EXAMPLE)
void doTCP(void)
cmx_reentrant {
   CHAR socket_no;
   PSOCKET_INFO socket_ptr;
   byte *data_ptr;
   word16 data_len;
   int status;

#if SERVER_MODE
   socket_no = mn_open(ip_dest_addr,ECHO_PORT,0,PASSIVE_OPEN,PROTO_TCP,\
                       STD_TYPE,data_buff,DATA_BUFF_LEN);
   data_ptr = PTR_NULL;
   data_len = 0;
#else
   socket_no = mn_open(ip_dest_addr,DEFAULT_PORT,ECHO_PORT,ACTIVE_OPEN,PROTO_TCP,\
                       STD_TYPE,data_buff,DATA_BUFF_LEN);
   data_ptr = TESTSTRING;
   data_len = strlen((char *)TESTSTRING);
#endif

   if (socket_no < 0)    /* did we open socket successfully? */
      EXIT(1);
```

```
   socket_ptr = MK_SOCKET_PTR(socket_no);       /* get pointer to the socket */
   while(1)
      {
#if (defined(__BORLANDC__) || defined(_MSC_VER))
     if (_bios_keybrd(_KEYBRD_READY))     /* check if ctrl-c pressed */
        {
        if ((_bios_keybrd(_KEYBRD_READ) & 0xff) == 3)
          {
          mn_close(socket_no);
          break;
          }
        }
#endif       /* #if (defined(__BORLANDC__) || defined(_MSC_VER)) */

      status = 0;
      /* mn_send sends the data and waits for an ACK. Some echo servers will
         return the ACK and the data in the same packet, so we need to handle
         that case.
      */
      if (data_ptr != PTR_NULL)
         {
         status = mn_send(socket_no,data_ptr,data_len);
         if (status > 0 && socket_ptr->recv_len > 0)
            {
            data_ptr = socket_ptr->recv_ptr;
            data_len = socket_ptr->recv_len;
            continue;
            }
         }
      if (status < 0 || socket_ptr->tcp_state == TCP_CLOSED)
         break;

      do
         {
         status = mn_recv(socket_no,data_buff,DATA_BUFF_LEN);
         }
      while (status == NEED_TO_LISTEN);
      if (status < 0 && status != SOCKET_TIMED_OUT)
         break;
      /* if we got something, send back what we got */
      if (status > 0)
         {
         data_ptr = socket_ptr->recv_ptr;
         data_len = socket_ptr->recv_len;
         }
#if PPP
      if (!(ppp_status.up))
         break;
#endif    /* PPP */
      }

   mn_abort(socket_no);
}
#endif       /* (EXAMPLE_TYPE == TCP_EXAMPLE) */

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

#if (EXAMPLE_TYPE == SMTP_EXAMPLE)
/* replace the email addresses below */
byte from[]     = "frodo@cmx.com";
byte to[]       = "bilbo@cmx.com";
byte subject[]  = "SMTP test";
```

```
byte message[]   = "This is the message body.\r\n";
byte attach[]    = "This is the attachment.\r\n";
byte fname[]     = "micronet.txt";

void doSMTP(void)
cmx_reentrant {
   CHAR socket_no;
   CHAR retval;
   SMTP_INFO_T mail_info;

   socket_no = mn_smtp_start_session(DEFAULT_PORT);

   if (socket_no >= 0)
      {
      /* send a message with an attachment */
      mail_info.from = from;
      mail_info.to = to;
      mail_info.subject = subject;
      mail_info.message = message;
      mail_info.attachment = attach;
      mail_info.filename = fname;

      retval = mn_smtp_send_mail(socket_no, &mail_info);

      if (retval < 0)
         {
         /* there was an error sending the message */
         ;
         }

      /* send a message without an attachment */
      mail_info.from = from;
      mail_info.to = to;
      mail_info.subject = subject;
      mail_info.message = message;
      mail_info.attachment = PTR_NULL;
      mail_info.filename = PTR_NULL;

      retval = mn_smtp_send_mail(socket_no, &mail_info);

      if (retval < 0)
         {
         /* there was an error sending the message */
         ;
         }

      mn_smtp_end_session(socket_no);
      }
}
#endif       /* (EXAMPLE_TYPE == SMTP_EXAMPLE) */

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

#if (EXAMPLE_TYPE == HTTP_EXAMPLE)
/* this function is called from a Web page by an HTTP POST request */
void lcd_msg_func(PSOCKET_INFO socket_ptr)
cmx_reentrant {
   /* we are expecting a variable called display with escaped string, e.g.
         display=hello%2C+world%21
      where '+' equals ' ' and %XX is a hexadecimal representation of a char.
   */

   /* msg_buff will have decoded value, if available */
```

```
   if (mn_http_find_value(BODYptr,(byte *)"display",msg_buff))
      {
#if !SERVER_SIDE_INCLUDES
      /* Just return good status. */
      socket_ptr->send_ptr = (byte *)HTTPStatus204;
      socket_ptr->send_len = STATUS_204_LEN;
#else
      /*    in this example we are always returning index.htm.
         we will cheat here and assume index.htm is the first entry
         in the virtual file directory.
      */
      socket_ptr->send_len = vf_dir[0].page_size;
      socket_ptr->send_ptr = vf_dir[0].page_ptr;
      mn_http_process_includes(socket_ptr);
#endif       /* #if !SERVER_SIDE_INCLUDES */
      }
   else
      {
         /* variable not found - issue a bad request error message */
      socket_ptr->send_ptr = (byte *)HTTPStatus400;
      socket_ptr->send_len = STATUS_400_LEN;
      }
}

#if (EXAMPLE_TYPE == HTTP_EXAMPLE)
/* this function is called from a Web page by an HTTP POST request */
void lcd_stop_func(PSOCKET_INFO socket_ptr)
cmx_reentrant {
   /* we are expecting a variable called display with escaped string, e.g.
         display=hello%2C+world%21
      where '+' equals ' ' and %XX is a hexadecimal representation of a char.
   */

   /* msg_buff will have decoded value, if available */
   if (mn_http_find_value(BODYptr,(byte *)"Stop",msg_buff))
      {
#if !SERVER_SIDE_INCLUDES
      /* Just return good status. */
      socket_ptr->send_ptr = (byte *)HTTPStatus204;
      socket_ptr->send_len = STATUS_204_LEN;
#else
      /*    in this example we are always returning index.htm.
         we will cheat here and assume index.htm is the first entry
         in the virtual file directory.
      */

      socket_ptr->send_len = vf_dir[0].page_size;
      socket_ptr->send_ptr = vf_dir[0].page_ptr;
      mn_http_process_includes(socket_ptr);
#endif       /* #if !SERVER_SIDE_INCLUDES */
      }
   else
      {
         /* variable not found - issue a bad request error message */
      socket_ptr->send_ptr = (byte *)HTTPStatus400;
      socket_ptr->send_len = STATUS_400_LEN;
      }
}
#if SERVER_SIDE_INCLUDES
/* this function is called from a Web page by a Server-Side-Include */
/* put current msg_buff in str and return number of bytes added to str */
word16 getMsg_func(byte *str)
cmx_reentrant {
```

```
    strcpy((char *)str,(char *)msg_buff);
    return ((word16)strlen((char *)msg_buff));
}
#endif        /* #if SERVER_SIDE_INCLUDES */
#endif    /* (EXAMPLE_TYPE == HTTP_EXAMPLE) */
```

7.1.2. Callback.c

```
/*********************************************************
Copyright (c) CMX Systems, Inc. 2001. All rights reserved
*********************************************************/

/*************************************************************
* CMX recommends making a copy of this file before changing *
* any of the routines below.                                *
*************************************************************/

#include "micronet.h"

/*************************************************************
   Change the ip_src_addr below to relect the IP address of
   your target. If the IP address of the destination is known
   replace ip_dest_addr with that address. If you are dialing
   into an ISP the source and destination addresses initially
   specified do not matter as they will be negotiated.
*************************************************************/

#if ETHERNET
/* #if (HTTP || FTP || SMTP) */
byte ip_dest_addr[IP_ADDR_LEN] = {216,233,5,32};
byte ip_src_addr[IP_ADDR_LEN]  = {216,233,5,26};
/* #else */
/* byte ip_dest_addr[IP_ADDR_LEN] = {209,67,233,68}; */
/* byte ip_src_addr[IP_ADDR_LEN]  = {209,67,233,67}; */
/* #endif */      /* HTTP || FTP */
#else
byte ip_dest_addr[IP_ADDR_LEN] = {192,6,94,5};
byte ip_src_addr[IP_ADDR_LEN]  = {192,6,94,2};
#endif      /* Ethernet */

#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] = { 0x00,0x00,0x00,0x12,0x34,0x56 };

/*************************************************************
   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 };
#endif      /* ETHERNET */

/* ---------------------------------------------------------------------- */
```

```
/* return number of bytes to send */
/* Called from mn_tcp_send and mn_udp_send */
word16 mn_app_get_send_size(PSOCKET_INFO socket_ptr)
cmx_reentrant {
  word16 bytes_to_send;

  bytes_to_send = (socket_ptr->send_ptr == PTR_NULL ? 0: socket_ptr->send_len);
#if (HTTP || FTP || SMTP)
  if ( (socket_ptr->socket_type & (HTTP_TYPE|SMTP_TYPE)) || \
       socket_ptr->src_port == FTP_DATA_PORT )
    {
    bytes_to_send = ((bytes_to_send > TCP_WINDOW) ? TCP_WINDOW : bytes_to_send);
    }
#endif
  return (bytes_to_send);
}

/* initialization before receiving a packet */
/* Called from mn_tcp_recv and mn_udp_recv */
/* num = number of bytes to be received */
void mn_app_init_recv(word16 num,PSOCKET_INFO socket_ptr)
cmx_reentrant {
#if (!(FTP && FTP_SERVER))
  num = num;
#endif     /* (!(FTP && FTP_SERVER)) */
  socket_ptr->recv_len = 0;

#if HTTP
  if (socket_ptr->socket_type & HTTP_TYPE)
    mn_http_init_recv();
  else
#endif
#if (FTP && FTP_SERVER)
  if (socket_ptr->socket_type & FTP_TYPE)
    mn_ftp_server_init_recv(num,socket_ptr);
  else
#endif
    {
    /* put your code here */
    ;
    }
}

/* process received byte. Called from mn_tcp_recv and mn_udp_recv */
void mn_app_recv_byte(byte c,PSOCKET_INFO socket_ptr)
cmx_reentrant {
#if HTTP
  if (socket_ptr->socket_type & HTTP_TYPE)
    mn_http_recv_byte(c);
  else
#endif
#if (FTP && FTP_SERVER)
  if (socket_ptr->src_port == FTP_CONTROL_PORT)
    mn_ftp_server_recv_byte(c);
  else
#endif
    {
    if (socket_ptr->recv_ptr != PTR_NULL)
      {
      /* copy the data into a buffer where we will deal with it later */
      if ((socket_ptr->recv_ptr+socket_ptr->recv_len) <= socket_ptr->recv_end)
        {
        *(socket_ptr->recv_ptr+socket_ptr->recv_len) = c;
```

```
            socket_ptr->recv_len++;
            }
          }
        }
}

/* Process received packet. Called from mn_tcp_recv and mn_udp_recv. */
void mn_app_process_packet(PSOCKET_INFO socket_ptr)
cmx_reentrant {
#if HTTP
  if (socket_ptr->socket_type & HTTP_TYPE)
    mn_http_process_packet(socket_ptr);
  else
#endif
#if (FTP && FTP_SERVER)
  if (socket_ptr->src_port == FTP_CONTROL_PORT)
    mn_ftp_server_process_packet(socket_ptr);
  else if (socket_ptr->src_port == FTP_DATA_PORT)
    mn_ftp_server_process_data(socket_ptr);
  else
#endif
    {
    /* remove the next line and put your code here */
    socket_ptr = socket_ptr;
    ;
    }
}

#if TCP
/*
  Do what needs to be done after successfully sending a packet.
  Called from mn_tcp_recv().
*/
void mn_app_send_complete(word16 data_len,PSOCKET_INFO socket_ptr)
cmx_reentrant {
  if (socket_ptr->send_len > 0)
    socket_ptr->send_len -= data_len;   /* subtract bytes already sent */

  /* move pointer to next spot to start sending if anything left to send,
    or set it to PTR_NULL otherwise.
  */
  if (socket_ptr->send_len > 0)
    socket_ptr->send_ptr += data_len;
  else
    socket_ptr->send_ptr = PTR_NULL;
}

#endif      /* #if TCP */

/* called from mn_recv */
CHAR mn_app_recv_idle(void)
cmx_reentrant {
  /* put your code here.

    return NEED_TO_EXIT if mn_recv must be exited immediately.
  */
  return (0);
}

#if HTTP
CHAR mn_app_http_server_idle(void)
cmx_reentrant {
  /* put your code here.
```

```
    return NEED_TO_SEND if data must be sent immediately.
  */
  return (0);
}
#endif    /* HTTP */

#if (FTP && FTP_SERVER)
CHAR mn_app_ftp_server_idle(void)
cmx_reentrant {
  /* put your code here.
    return NEED_TO_SEND if data must be sent immediately.
  */
  return (0);
}
#endif    /* #if (FTP && FTP_SERVER) */
```

7.1.3. nec_hw.c

```
/*=========================================================================
** PROJECT     = API for CMX Implementation
** MODULE      = NEC_hw.c
** SHORT DESC.  = Hardware setup for CMX stack
** DEVICE      = uPD78F0066(specify change in project options to choose different processor)
** VERSION     = 1.1
** DATE        = 04.01.2002
** LAST CHANGE  = -
** =========================================================================
** Description:  API Implementation
**             The 78K - Test It! demo board implements CMX Internet connectivity stack
**             8-bit timer required stack tick, UART required for connection
**             This file can be modified to change hardware requirements for
**             CMX stack.
**
** =========================================================================
** Environment:  Device:       uPD78F0066
**             Assembler:     A78000       Version     V3.33A
**             C-Compiler:    ICC78000     Version     V3.33A
**             Linker:        XLINK        Version     V4.52J
** =========================================================================
** By:          NEC Electronics (UK) Ltd
**             Cygnus House
**             Sunrise Parkway
**             Linford Wood
**             Milton Keynes UK MK14 6NP
** =========================================================================
Changes:     Incorporate V2.15f into API. Update Interrupt buffer handling
** =========================================================================
*/

#include "micronet.h"

volatile word16 timer_tick;                    /* Restart timer */

#define KEEP_ERROR_COUNT          0                    /* set to 1 to count errors */
#if (KEEP_ERROR_COUNT)
volatile word16 uart_errors;
#endif

extern void vTimer1(void);

/* ------------------------------------------------------------------- */
#if (PPP || SLIP)
```

```
void mn_uart_init(void)
cmx_reentrant
        {
        DISABLE_INTERRUPTS;
        init_io_buffs();                        /* do not remove this function call */
        /* put uart init code here */
#if   (defined(D78076))
        PM70 = 1;                   //Set RXD to input mode
        PM71 = 0;                   //Set TXD to output mode
        P7.1 = 1;
#endif
#if   (defined(D78F0034)|| defined(D78F0078))
        PM23 = 1;                   //Set RXD to input mode
        PM24 = 0;                   //Set TXD to output mode
#endif
#if   (defined(D78F0066))
        PM73 = 1;                   //Set RXD to input mode
        PM72 = 0;                   //Set TXD to output mode
#endif

#if   (defined(D78F0828))
      PM62 = 1;                   //Set RXD to input mode
      PM63 = 0;               //Set TXD to output mode
#endif

#if ((defined(D78F0034)) || (defined(D78F0066)) || (defined(D78F0078)) || (defined(D78F0828)))
        ASIM0 = 0x08;                   /* disable uart No Parity 8bit 1stop*/

        BRGC0 = Baud_38400_4_194304MHz;

      SRIF0  = FALSE;               // Clear Interrupt request reception complete
      STIF0  = FALSE;               // Clear Interrupt request transmision complete
      SRMK0  = FALSE;                 // Enable Interrupt reception complete
      STMK0  = FALSE;                 // Enable Interrupt transmision complete
        ASIM0 = 0xC8;                  /* Enable UART receive and transmit */
#endif
#if (defined(D78076))
      ASIM = 0x09;                  /* disable uart No Parity 8bit 1stop*/

      BRGC = Baud_38400_5MHz;           /* 38400 @ 5Mhz */

      SRIF  = FALSE;                // Clear Interrupt request reception complete
      STIF  =  FALSE;               // Clear Interrupt request transmision complete
      SRMK  = FALSE;                  // Enable Interrupt reception complete
      STMK  =  FALSE;                 // Enable Interrupt transmision complete
        ASIM = 0xC9;                   /* Enable UART receive and transmit */
#endif

        ENABLE_INTERRUPTS;
        }
#endif          /* (PPP || SLIP) */

/* put uart isr(s) here */
/* place service routine for receive buffer here */
#if ((defined(D78F0034)) || (defined(D78F0066)) || (defined(D78F0078)) || (defined(D78F0828)))
interrupt [INTSR0_vect] using[3] void vUARTReceive(void)
{
 //     P5.3 = 0;
/*        if (recv_count <= RECV_BUFF_SIZE) */
        if (recv_count < (RECV_BUFF_SIZE-1))          //Fix for buffer management problems
                {
                   RTS = 0;
                   *recv_in_ptr = RXB0;
```

```
                        ++recv_count;
                        ++recv_in_ptr;
                        if (recv_in_ptr > &recv_buff[RECV_BUFF_SIZE-1])
                                recv_in_ptr = &recv_buff[0];
                    }
        else
        {
           RTS = 1;
        }
//          P5.3 = 1;
}
#endif

#if (defined(D78076))
interrupt [INTSR_vect] using[3] void vUARTReceive(void)
{

/*          if (recv_count <= RECV_BUFF_SIZE) */
            if (recv_count < (RECV_BUFF_SIZE-1))            //Fix for buffer management problems
                    {
                        RTS = 0;
                        *recv_in_ptr = RXB;
                        ++recv_count;
                        if (recv_in_ptr > &recv_buff[RECV_BUFF_SIZE-1])
                                recv_in_ptr = &recv_buff[0];
                    }
        else
        {
           RTS = 1;
            }
}
#endif
#if ((defined(D78F0034)) || (defined(D78F0066)) || (defined(D78F0078)) || (defined(D78F0828)))
interrupt [INTSER0_vect] using[3] void eri0(void)
{

//byte temp;
#if (KEEP_ERROR_COUNT)
            uart_errors++;
#endif

            if ( ASIS0 & 0x01 )         ;                               /* overrun error */
//                      temp = RXB0                 /* if errors must read RXB to clear rubbish data */
            else if ( ASIS0 & 0x04 );           /* parity error */

            else if ( ASIS0 & 0x02 );           /* framing error */
            P5 = 0;                             /* Visual Test for errors */
}
#endif

/* place service routine for transmit buffer here */
#if ((defined(D78F0034)) || (defined(D78F0066)) || (defined(D78F0078)) || (defined(D78F0828)))
interrupt [INTST0_vect] using[2] void vUARTTransmit(void)
            {
//        P5.7 = 0;
            if (send_out_ptr != send_in_ptr)
                    {
                      TXS0 = *send_out_ptr;

                ++send_out_ptr;
                    }
                    else
                    {
```

```
                    xmit_busy = 0;
                    STMK0 = 1;
//              P5.7 = 1;
                }
//      P5.7 = 1;
        }                   /* (PPP || SLIP) */
#endif
#if (defined(D78076))
interrupt [INTST_vect] using[2] void vUARTTransmit(void)
        {
        if (send_out_ptr != send_in_ptr)
                {
                  TXS = *send_out_ptr;

            ++send_out_ptr;
          }
              else
                {
                   xmit_busy = 0;
                }
        }                   /* (PPP || SLIP) */
#endif


/* put timer isr here */
#if ((defined(D78F0034)) || (defined(D78F0066)) || (defined(D78F0078)) || (defined(D78F0828)))
interrupt [INTTM50_vect] using[1] void vTimer(void)
#endif
#if (defined(D78076))
interrupt [INTTM1_vect] using[1] void vTimer(void)
#endif
{
        ++timer_tick;
#if DHCP
                mn_dhcp_update_timer();
#endif                              /* DHCP */
#if (ARP && ARP_TIMEOUT)
                mn_arp_update_timer();
#endif          /* (ARP && ARP_TIMEOUT) */

}

void mn_timer_init(void)
cmx_reentrant
        {
        DISABLE_INTERRUPTS;
        /* put timer init code here Must achieve 10ms count for operation */
#if (defined(D78076))
      TMC1    = 0x00;             // Stop timer operation
      TCL1    = 0xFD;             // Set timer clock : fx/2^9 (5MHz -> 9800 Hz)
      CR10    = 0x31;             // Timer1 compare value ->  10.00ms intervall
      TOC1    = 0x03;             // Set to output mode
      PM31    = 0;                //Set to output Mode
      TMIF1   = FALSE;              // Clear interrupt request Bit Timer1
      TMMK1   = FALSE;                // Enable Timer1 Interrupt
      TMC1    = 0x01;             // output, no cascade mode
#endif

#if (defined(D78F0828))
      TMC50   = 0x00;              // Stop timer operation
      TCL50   = 0x05;             // Set timer clock :   fx/2^8 (4.194304 MHz -> 16384 Hz)
      CR50    = 0x52;             // Timer50 compare value ->  10.01ms intervall
      TMC50   = 0x03;              // output, no cascade mode
```

```
#endif

#if ((defined(D78F0034)) || (defined(D78F0066)) || (defined(D78F0078)))
      TMC50   = 0x00;                 // Stop timer operation
      TCL50   = 0x06;                 // Set timer clock :   fx/2^8 (4.194304 MHz -> 16384 Hz)
      CR50    = 0x52;                 // Timer50 compare value ->  10.01ms intervall
      TMC50   = 0x03;                 // output, no cascade mode
#endif

#if ((defined(D78F0034))|| (defined(D78F0078)))
      PM72 = 0;                       //Set to output Mode
      TMIF50  = FALSE;                // Clear interrupt request Bit Timer50
      TMMK50  = FALSE;                // Enable Timer50 Interrupt
      TMC50.7 = 1;
#endif

#if (defined(D78F0066))
      PM22 = 0;                       //Set to output Mode
      TMIF50  = FALSE;                // Clear interrupt request Bit Timer50
      TMMK50  = FALSE;                // Enable Timer50 Interrupt
      TMC50.7 = 1;
#endif

#if (defined(D78F0828))
      PM34 = 0;                       //Set to output Mode
      TMIF50  = FALSE;                // Clear interrupt request Bit Timer50
      TMMK50  = FALSE;                // Enable Timer50 Interrupt
      TMC50.7 = 1;
#endif

        timer_tick = 0;
        ENABLE_INTERRUPTS;
}
```

7.1.4. init.c

```
/*===========================================================================
** PROJECT     = 78K0 Demonstration Board
** MODULE      = init.c
** SHORT DESC. = -
** DEVICE      = uPD78F0066
** VERSION     = 1.0
** DATE        = 27.11.2001
** LAST CHANGE = -
** ===========================================================================
** Description: Initialization of Registers and Bit Variables
** ===========================================================================
** Environment: Device:       uPD78F0066
**              Assembler:    A78000      Version   V3.33A
**              C-Compiler:   ICC78000    Version   V3.33A
**              Linker:       XLINK       Version   V4.52J
** ===========================================================================
** By:          NEC Electronics (UK) Ltd
**              Cygnus House
**              Sunrise Parkway
**              Linford Wood
**              Milton Keynes UK MK14 6NP
** ===========================================================================
Changes:
** ===========================================================================
*/
```

```
//---------------------------------------------------------------------------
// Include files
//---------------------------------------------------------------------------
#include <in78000.h>
#if (defined(D78F0034))
  #include <df0034.h>                       /* change to appropriate file */
#endif
#if (defined(D78F0066))
  #include <df0066.h>                       /* change to appropriate file */
#endif
#if (defined(D78F0078))
  #include <df0078.h>                       /* change to appropriate file */
#endif
#if (defined(D78F0828))
  #include <df0828.h>                 /* change to appropriate file */
#endif
#if (defined(D78076))
  #include <d076.h>                 /* change to appropriate file */
#endif

#include "micronet.h"

//---------------------------------------------------------------------------
// Global variables
//---------------------------------------------------------------------------
extern unsigned int Compiler_Version_Number;

/* ============================================================================
**  Module name: vHardwareInit
**
**  Description:
**          This module is to initialize some peripheral hardware.
**
**  Operation:
**          Sets the clock generator, the port modes and output latches
**          and initializes the interrupts.
** ============================================================================
*/
void vHardwareInit(void)            // Hardware inizialization
{
#if (defined(D78F0066))
// clock generator setting
//-------------------------
   PCC  = 0x00;                  // Use high speed mode
   OSTS = 0x02;                   // 2^15/fx ms wait after STOP release by interrupt
   IMS  = 0xCC;                  // Select 1024 Byte RAM and 48k Byte ROM
   IXS  = 0x04;                 // Select 4096 Byte RAM

// port setting
//-------------
   P0    = 0x00;                // Set output latch to 0
   P2    = 0x00;                // Set output latch to 0
   P3    = 0x00;                // Set output latch to 0
   P4    = 0x00;                // Set output latch to 0
   P5    = 0xff;               // Set output latch to 0xff
   P6    = 0x00;                // Set output latch to 0
   P7    = 0x01;                // Set output latch to 0x01 (enable RS232 driver)
   P8    = 0x00;                // Set output latch to 0
   P8    = 0x00;                // Set output latch to 0
   PU7   = 0x08;                // Enable internal pull-up resistor P7.3
   PM0   = 0xFF;                 // Port 0 =  input (8-bits)
   PM2   = 0xFF;                 // Port 2 =  input (8-bits)
   PM3   = 0xFF;                 // Port 3 =  input (8-bits)
```

```
    PM4    = 0x00;                  // Port 4 =  input (8-bits)
    PM5    = 0x00;                  // Port 5 = output (8-bits)
    PM6    = 0xFF;                  // Port 6 =  input (4-bits)
    PM7    = 0x88;                  // Port 7.7,7.3 input, other output
    PM8    = 0xE0;                  // Port 8 = output (5 bits)
    PM9    = 0xFF;                  // Port 9 =  input (3-bits)

// interrupt setting
//------------------

    IF0L   = 0x00;                 // Reset all interrupt request bits
    IF0H   = 0x00;
    IF1L   = 0x00;
    MK0L   = 0xFF;                  // Disable all interrupts
    MK0H   = 0xFF;
    MK1L   = 0xFF;
    PR0L   = 0xFF;                  // Set all interrupts to low priority
    PR0H   = 0xFF;
    PR1L   = 0xFF;

// Compiler Version Number setting
// --------------------

    Compiler_Version_Number = __VER__;          //Must be left in code so Micronet will work with current
compiler version
#endif

#if (defined(D78076))
// clock generator setting
//-------------------------
    PCC  = 0x00;                   // Use high speed mode
    OSTS = 0x02;                    // 2^15/fx ms wait after STOP release by interrupt
    OSMS = 0x01;                    // Scalar not used
    IMS  = 0xCC;                   // Select 1024 Byte RAM and 48k Byte ROM
    IXS  = 0x0A;                   // Select 1024 Byte RAM
#endif
}

/* ============================================================================
**  Module name: vSoftwareInit
**
**  Description:
**              This module is to initialize the bit variables
**
**  Operation:
**              -
** ============================================================================
*/
void vSoftwareInit(void)
{
}
```

7.2. Examples

7.2.1. Electronic Dice and A/D Input

The example below is based on the original 78K – Test It! Electronic Dice Program. The main changes to the electronic dice example allows the user to use a Web browser to to initiate and display the result of the electronic dice. In addition to this a second Web page and application was added to show how to add more than one application to CMX-Micronet.

Therefore it is possible to use the 78K – Test It! User Manual for the flow and resources used for this example. Additions to this example include a conversion.c to convert the A/D value and dice result to ASCII to allow insertion via SSI to the HTML page.

7.2.1.1. Example.c

```
/*************************************************************
Copyright (c) CMX Systems, Inc. 2001. All rights reserved
*************************************************************/

#include "micronet.h"
#include "User_config.h"

#define SERVER_MODE       1    /* set to 1 if a server, or 0 if a client */
#define MODEM_MODE      ANSWER_MODE    /* set to DIAL_MODE or ANSWER_MODE */

#define UDP_EXAMPLE       0
#define TCP_EXAMPLE       1
#define HTTP_EXAMPLE      2
#define FTP_EXAMPLE       3
#define SMTP_EXAMPLE      4

/*************************************************************
   To setup the example program for the UDP echo example
   make sure UDP is selected in config.h then change the
   line of code below to:
   #define EXAMPLE_TYPE    UDP_EXAMPLE

   To setup the example program for the TCP echo example
   make sure TCP is selected in config.h then change the
   line of code below to:
   #define EXAMPLE_TYPE    TCP_EXAMPLE

   To setup the example program for the HTTP server example
   make sure TCP and HTTP are selected in config.h then
   change the line of code below to:
   #define EXAMPLE_TYPE    HTTP_EXAMPLE

   To setup the example program for the FTP server example
   make sure TCP and FTP are selected in config.h then
   change the line of code below to:
   #define EXAMPLE_TYPE    FTP_EXAMPLE

   To setup the example program for the SMTP server example
   make sure TCP and SMTP are selected in config.h then
   change the line of code below to:
   #define EXAMPLE_TYPE    SMTP_EXAMPLE
*************************************************************/
#define EXAMPLE_TYPE    HTTP_EXAMPLE

#if !UDP && EXAMPLE_TYPE == UDP_EXAMPLE
#error UDP example selected without UDP
#endif
#if !TCP && EXAMPLE_TYPE == TCP_EXAMPLE
#error TCP example selected without TCP
#endif
#if !HTTP && EXAMPLE_TYPE == HTTP_EXAMPLE
#error HTTP example selected without HTTP
#endif
#if !FTP && EXAMPLE_TYPE == FTP_EXAMPLE
#error FTP example selected without FTP
#endif
```

```
#if !SMTP && EXAMPLE_TYPE == SMTP_EXAMPLE
#error SMTP example selected without SMTP
#endif


#if ((EXAMPLE_TYPE == HTTP_EXAMPLE) || (EXAMPLE_TYPE == FTP_EXAMPLE))
/* put #includes for Web pages here */
#include "index.h"
#include "page2.h"
#include "necem.h"
#include "table.h"
#endif    /* ((EXAMPLE_TYPE == HTTP_EXAMPLE) || (EXAMPLE_TYPE == FTP_EXAMPLE)) */


#if (EXAMPLE_TYPE == HTTP_EXAMPLE)
#undef SERVER_MODE
#define SERVER_MODE    1

void lcd_msg_func(PSOCKET_INFO) cmx_reentrant;
void lcd_stop_func(PSOCKET_INFO) cmx_reentrant;
#if SERVER_SIDE_INCLUDES
word16 getMsg_func(byte *) cmx_reentrant;
word16 dice_func(byte *) cmx_reentrant;
word16 getAtoD_func0(byte *) cmx_reentrant;
word16 getAtoD_func2(byte *) cmx_reentrant;
word16 getAtoD_func3(byte *) cmx_reentrant;
word16 Version_func(byte *) cmx_reentrant;
word16 Code_func(byte *) cmx_reentrant;
#endif


#elif (EXAMPLE_TYPE == FTP_EXAMPLE)
/* for now ftp is only server, this will change later */
#undef SERVER_MODE
#define SERVER_MODE    1

#elif (EXAMPLE_TYPE == SMTP_EXAMPLE)
/* smtp is only client */
#undef SERVER_MODE
#define SERVER_MODE    0
#else
static byte TESTSTRING[]= "THIS IS A TEST STRING TO ECHO";
#define DATA_BUFF_LEN    40
byte data_buff[DATA_BUFF_LEN];
#endif    /* (EXAMPLE_TYPE == HTTP_EXAMPLE) */

#if (defined(__BORLANDC__) || defined(_MSC_VER))
extern void (__interrupt __far *old1b)();        /* to hold old int 0x1b handler */
#endif

#if (EXAMPLE_TYPE == UDP_EXAMPLE)
void doUDP(void) cmx_reentrant;
#elif (EXAMPLE_TYPE == TCP_EXAMPLE)
void doTCP(void) cmx_reentrant;
#elif (EXAMPLE_TYPE == SMTP_EXAMPLE)
void doSMTP(void) cmx_reentrant;
#endif        /* (EXAMPLE_TYPE == UDP_EXAMPLE) */

#if BOOTP
   BOOTP_INFO_T bp;
#endif        /* BOOTP */

#if TFTP
```

```
#define TESTLEN    64
byte testBuffer[TESTLEN];
long length;
#endif        /* BOOTP */

extern byte AtoD[4];
extern byte AtoD_buff[8];
static byte msg_buff[17];
extern bit bKey1Flag;
extern saddr unsigned char ucThrow;
const unsigned char Welcome[] =
"Welcome to NEC\r\n";

extern byte ASCII_DEC(byte a);
extern void DEC_ASCII(byte a, byte b, byte c);          // DEC-ASCII conversion
extern void dice(void);
#if (defined(D78F0066))
  extern void AtoD_Values(void);
#endif

/* ---------------------------------------------------------------------- */
void main(void)
{
#if MODEM
    byte connect_mode;
#endif

#if PPP
    byte open_mode;
#endif

#if MODEM
#if (MODEM_MODE == ANSWER_MODE)
    connect_mode = ANSWER_MODE;
#else
    connect_mode = DIAL_MODE;
#endif
#endif    /* MODEM */

#if PPP
#if SERVER_MODE
    open_mode = PASSIVE_OPEN;
#else
    open_mode = ACTIVE_OPEN;
#endif
#endif        /* #if PPP */

#if (defined(POL78K0) || defined(CMX78K0))
    /*  put your hardware initialisation routines here for all peripherals and sfr's Except Timer and UART used
by stack*/
    vHardwareInit();
    vSoftwareInit();                    // Variable Initialization

#endif

    /* call mn_init before using any other MicroNet API functions */
    mn_init();

#if (defined(__BORLANDC__) || defined(_MSC_VER))
    old1b = getvector(0x1b);
    setvector(0x1b,onbreak);        /* stop ctrl-break */
    setvector(0x23,onbreak);
    atexit(cmx_restore);
```

```
#endif       /* #if (defined(__BORLANDC__) || defined(_MSC_VER)) */

#if ((EXAMPLE_TYPE == HTTP_EXAMPLE) || (EXAMPLE_TYPE == FTP_EXAMPLE))
   /* add Web pages to virtual file system. The main page MUST be first */
   mn_vf_set_entry((byte *)"index.htm", INDEX_SIZE, (void *)index_htm,VF_PTYPE_STATIC);
   mn_vf_set_entry((byte *)"page2.htm", PAGE2_SIZE, (void *)page2_htm,VF_PTYPE_STATIC);
   mn_vf_set_entry((byte *)"necem.gif", NECEM_SIZE, (void *)necem_gif,VF_PTYPE_STATIC);
   mn_vf_set_entry((byte *)"table.gif", TABLE_SIZE, (void *)table_gif,VF_PTYPE_STATIC);


#if (EXAMPLE_TYPE == HTTP_EXAMPLE)
   /* add post functions to be used with forms */
   strcpy((char *)msg_buff,(char *)Welcome);
   mn_pf_set_entry((byte *)"lcd_msg", lcd_msg_func);
   mn_pf_set_entry((byte *)"lcd_stop", lcd_stop_func);

#if SERVER_SIDE_INCLUDES
   /* add any get functions (server-side-includes) here */
  // AtoD_Values();
   mn_gf_set_entry((byte *)"getMsg", getMsg_func);
   mn_gf_set_entry((byte *)"dice", dice_func);
  mn_gf_set_entry((byte *)"getAtoD0", getAtoD_func0);
   mn_gf_set_entry((byte *)"getAtoD2", getAtoD_func2);
   mn_gf_set_entry((byte *)"getAtoD3", getAtoD_func3);
   mn_gf_set_entry((byte *)"Version", Version_func);
   mn_gf_set_entry((byte *)"Code", Code_func);
#endif       /* SERVER_SIDE_INCLUDES */
#endif       /* (EXAMPLE_TYPE == HTTP_EXAMPLE) */
#endif    /* ((EXAMPLE_TYPE == HTTP_EXAMPLE) || (EXAMPLE_TYPE == FTP_EXAMPLE)) */

while(1)
{
#if MODEM
   if (mn_modem_connect(connect_mode) < 0)
      EXIT(1);
#endif       /* MODEM */

#if PPP
#if USE_PAP
  mn_ppp_add_pap_user("cmx","password");
#endif     /* USE_PAP */

  if (!mn_ppp_open(open_mode))
      {
#if MODEM
      mn_modem_disconnect();
#endif    /* MODEM */
      EXIT(1);
      }
#endif       /* #if PPP */

#if DHCP
#define LEASE_TIME    1000
   if (mn_dhcp_start(PTR_NULL, LEASE_TIME) == 1)
      {
#if TFTP
      length = mn_tftp_get_file(dhcp_lease.server_id, dhcp_info.file, \
            testBuffer, TESTLEN);
         if (length < 0)       /* TFTP error */
            EXIT(1);
#endif       /* TFTP */
      }
   else          /* dhcp error */
```

```
        EXIT(1);
#elif BOOTP
    if ( mn_bootp(PTR_NULL,&bp) > 0)
        {
        /* if the #define BOOTP_REQUEST_IP was set to 0, should check here if
            bp.yiaddr is different than ip_src_addr.
        */
#if TFTP
        length = mn_tftp_get_file(bp.siaddr, bp.file, testBuffer, TESTLEN);
         if (length < 0)        /* TFTP error */
            EXIT(1);
#endif        /* TFTP */
        }
    else                        /* BOOTP error */
        EXIT(1);
#elif TFTP                /* tftp without bootp or dhcp */
    /* replace boot.mn with the file you wish to retrieve */
    length = mn_tftp_get_file(ip_dest_addr, (byte *)"boot.mn", testBuffer, TESTLEN);
     if (length < 0)        /* TFTP error */
        EXIT(1);
#endif        /* BOOTP */

#if (EXAMPLE_TYPE == UDP_EXAMPLE)
    doUDP();
#elif (EXAMPLE_TYPE == TCP_EXAMPLE)
    doTCP();
#elif (EXAMPLE_TYPE == HTTP_EXAMPLE)
    mn_http_server();        /* see http.c */
#elif (EXAMPLE_TYPE == FTP_EXAMPLE)
    mn_ftp_server();        /* see ftpservr.c */
#elif (EXAMPLE_TYPE == SMTP_EXAMPLE)
    doSMTP();
#endif        /* (EXAMPLE_TYPE == UDP_EXAMPLE) */

#if DHCP
    mn_dhcp_release();
#elif PPP
    mn_ppp_close();
#endif        /* #if PPP */

#if MODEM
    mn_modem_disconnect();
#endif    /* MODEM */

}
    EXIT(0);
}

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

#if (EXAMPLE_TYPE == UDP_EXAMPLE)
void doUDP(void)
cmx_reentrant {
    CHAR socket_no;
    PSOCKET_INFO socket_ptr;
    byte *data_ptr;
    word16 data_len;
    int status;

#if SERVER_MODE
    socket_no = mn_open(ip_dest_addr,ECHO_PORT,0,PASSIVE_OPEN,PROTO_UDP,\
                        STD_TYPE,data_buff,DATA_BUFF_LEN);
    data_ptr = PTR_NULL;
```

```
    data_len = 0;
#else       /* client mode */
    socket_no = mn_open(ip_dest_addr,DEFAULT_PORT,ECHO_PORT,ACTIVE_OPEN,PROTO_UDP,\
                        STD_TYPE,data_buff,DATA_BUFF_LEN);
    data_ptr = TESTSTRING;
    data_len = strlen((char *)TESTSTRING);
#endif

    if (socket_no < 0)    /* did we open socket successfully? */
        EXIT(1);

    socket_ptr = MK_SOCKET_PTR(socket_no);        /* get pointer to the socket */

    while(1)
        {
#if (defined(__BORLANDC__) || defined(_MSC_VER))
      if (_bios_keybrd(_KEYBRD_READY))     /* check if ctrl-c pressed */
          {
          if ((_bios_keybrd(_KEYBRD_READ) & 0xff) == 3)
            {
            mn_close(socket_no);
            break;
            }
          }
#endif       /* #if (defined(__BORLANDC__) || defined(_MSC_VER)) */

        status = 0;
        if (data_ptr != PTR_NULL)
            status = mn_send(socket_no,data_ptr,data_len);
        if (status < 0)
                {
                data_len = 0;
            break;
                }
        status = mn_recv(socket_no,data_buff,DATA_BUFF_LEN);
        if (status < 0 && status != SOCKET_TIMED_OUT)
                {
                data_len = 0;
            break;
                }
        /* if we got something, send back what we got */
        if (status > 0)
            {
            data_ptr = socket_ptr->recv_ptr;
            data_len = socket_ptr->recv_len;
            }

#if PPP
        if (!(ppp_status.up))
            break;
#endif    /* PPP */
        }

    mn_abort(socket_no);
}
#endif

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

#if (EXAMPLE_TYPE == TCP_EXAMPLE)
void doTCP(void)
cmx_reentrant {
    CHAR socket_no;
```

```
   PSOCKET_INFO socket_ptr;
   byte *data_ptr;
   word16 data_len;
   int status;

#if SERVER_MODE
   socket_no = mn_open(ip_dest_addr,ECHO_PORT,0,PASSIVE_OPEN,PROTO_TCP,\
                       STD_TYPE,data_buff,DATA_BUFF_LEN);
   data_ptr = PTR_NULL;
   data_len = 0;
#else
   socket_no = mn_open(ip_dest_addr,DEFAULT_PORT,ECHO_PORT,ACTIVE_OPEN,PROTO_TCP,\
                       STD_TYPE,data_buff,DATA_BUFF_LEN);
   data_ptr = TESTSTRING;
   data_len = strlen((char *)TESTSTRING);
#endif

   if (socket_no < 0)    /* did we open socket successfully? */
      EXIT(1);

   socket_ptr = MK_SOCKET_PTR(socket_no);       /* get pointer to the socket */
   while(1)
      {
#if (defined(__BORLANDC__) || defined(_MSC_VER))
      if (_bios_keybrd(_KEYBRD_READY))     /* check if ctrl-c pressed */
         {
         if ((_bios_keybrd(_KEYBRD_READ) & 0xff) == 3)
           {
           mn_close(socket_no);
           break;
           }
         }
#endif       /* #if (defined(__BORLANDC__) || defined(_MSC_VER)) */

      status = 0;
      /* mn_send sends the data and waits for an ACK. Some echo servers will
         return the ACK and the data in the same packet, so we need to handle
         that case.
      */
      if (data_ptr != PTR_NULL)
         {
         status = mn_send(socket_no,data_ptr,data_len);
         if (status > 0 && socket_ptr->recv_len > 0)
            {
            data_ptr = socket_ptr->recv_ptr;
            data_len = socket_ptr->recv_len;
            continue;
            }
         }
      if (status < 0 || socket_ptr->tcp_state == TCP_CLOSED)
         break;

      do
         {
         status = mn_recv(socket_no,data_buff,DATA_BUFF_LEN);
         }
      while (status == NEED_TO_LISTEN);
      if (status < 0 && status != SOCKET_TIMED_OUT)
         break;
      /* if we got something, send back what we got */
      if (status > 0)
         {
         data_ptr = socket_ptr->recv_ptr;
```

```
            data_len = socket_ptr->recv_len;
            }
#if PPP
        if (!(ppp_status.up))
            break;
#endif    /* PPP */
        }

    mn_abort(socket_no);
}
#endif        /* (EXAMPLE_TYPE == TCP_EXAMPLE) */

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

#if (EXAMPLE_TYPE == SMTP_EXAMPLE)
/* replace the email addresses below */
byte from[]      = "frodo@cmx.com";
byte to[]        = "bilbo@cmx.com";
byte subject[]   = "SMTP test";
byte message[]   = "This is the message body.\r\n";
byte attach[]    = "This is the attachment.\r\n";
byte fname[]     = "micronet.txt";

void doSMTP(void)
cmx_reentrant {
    CHAR socket_no;
    CHAR retval;
    SMTP_INFO_T mail_info;

    socket_no = mn_smtp_start_session(DEFAULT_PORT);

    if (socket_no >= 0)
        {
        /* send a message with an attachment */
        mail_info.from = from;
        mail_info.to = to;
        mail_info.subject = subject;
        mail_info.message = message;
        mail_info.attachment = attach;
        mail_info.filename = fname;

        retval = mn_smtp_send_mail(socket_no, &mail_info);

        if (retval < 0)
            {
            /* there was an error sending the message */
            ;
            }

        /* send a message without an attachment */
        mail_info.from = from;
        mail_info.to = to;
        mail_info.subject = subject;
        mail_info.message = message;
        mail_info.attachment = PTR_NULL;
        mail_info.filename = PTR_NULL;

        retval = mn_smtp_send_mail(socket_no, &mail_info);

        if (retval < 0)
            {
            /* there was an error sending the message */
            ;
```

```
        }

      mn_smtp_end_session(socket_no);
      }
}
#endif      /* (EXAMPLE_TYPE == SMTP_EXAMPLE) */

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

#if (EXAMPLE_TYPE == HTTP_EXAMPLE)
/* this function is called from a Web page by an HTTP POST request */
void lcd_msg_func(PSOCKET_INFO socket_ptr)
cmx_reentrant {
   /* we are expecting a variable called display with escaped string, e.g.
        display=hello%2C+world%21
      where '+' equals ' ' and %XX is a hexadecimal representation of a char.
   */

   /* msg_buff will have decoded value, if available */
   if (mn_http_find_value(BODYptr,(byte *)"display",msg_buff))
      {
#if !SERVER_SIDE_INCLUDES
      /* Just return good status. */
      socket_ptr->send_ptr = (byte *)HTTPStatus204;
      socket_ptr->send_len = STATUS_204_LEN;
#else
      /*    in this example we are always returning index.htm.
         we will cheat here and assume index.htm is the first entry
         in the virtual file directory.
      */
          //P5 = ASCII_DEC(*msg_buff);
          bKey1Flag = 1 ;              //Allow Web page to update dice throw
      socket_ptr->send_len = vf_dir[0].page_size;
      socket_ptr->send_ptr = vf_dir[0].page_ptr;
      mn_http_process_includes(socket_ptr);
#endif      /* #if !SERVER_SIDE_INCLUDES */
      }
   else
      {
         /* variable not found - issue a bad request error message */
      socket_ptr->send_ptr = (byte *)HTTPStatus400;
      socket_ptr->send_len = STATUS_400_LEN;
      }
}

#if (EXAMPLE_TYPE == HTTP_EXAMPLE)
/* this function is called from a Web page by an HTTP POST request */
void lcd_stop_func(PSOCKET_INFO socket_ptr)
cmx_reentrant {
   /* we are expecting a variable called display with escaped string, e.g.
        display=hello%2C+world%21
      where '+' equals ' ' and %XX is a hexadecimal representation of a char.
   */

   /* msg_buff will have decoded value, if available */
   if (mn_http_find_value(BODYptr,(byte *)"Stop",msg_buff))
      {
#if !SERVER_SIDE_INCLUDES
      /* Just return good status. */
      socket_ptr->send_ptr = (byte *)HTTPStatus204;
      socket_ptr->send_len = STATUS_204_LEN;
#else
      /*    in this example we are always returning index.htm.
```

```
            we will cheat here and assume index.htm is the first entry
            in the virtual file directory.
        */
             ucThrow = 0;                                        //added to reset dice application.
        socket_ptr->send_len = vf_dir[0].page_size;
        socket_ptr->send_ptr = vf_dir[0].page_ptr;
        mn_http_process_includes(socket_ptr);
#endif      /* #if !SERVER_SIDE_INCLUDES */
        }
    else
        {
            /* variable not found - issue a bad request error message */
        socket_ptr->send_ptr = (byte *)HTTPStatus400;
        socket_ptr->send_len = STATUS_400_LEN;
        }
}
#if SERVER_SIDE_INCLUDES
/* this function is called from a Web page by a Server-Side-Include */
/* put current msg_buff in str and return number of bytes added to str */
word16 getMsg_func(byte *str)
cmx_reentrant {
    strcpy((char *)str,(char *)msg_buff);
    return ((word16)strlen((char *)msg_buff));
}
word16 dice_func(byte *str)
cmx_reentrant {
    dice();
    DEC_ASCII(ucThrow,1,0);
    strcpy((char *)str,(char *)AtoD);
    return ((word16)strlen((char *)AtoD));
}
word16 getAtoD_func0(byte *str)
cmx_reentrant {
    AtoD_Values();
    DEC_ASCII(AtoD_buff[0],3,1);
    strcpy((char *)str,(char *)AtoD);
    return ((word16)strlen((char *)AtoD));
}

word16 getAtoD_func2(byte *str)
cmx_reentrant {

    DEC_ASCII(AtoD_buff[2],3,1);
    strcpy((char *)str,(char *)AtoD);
    return ((word16)strlen((char *)AtoD));
}
word16 getAtoD_func3(byte *str)
cmx_reentrant {

    DEC_ASCII(AtoD_buff[3],3,1);
    strcpy((char *)str,(char *)AtoD);
    return ((word16)strlen((char *)AtoD));
}
word16 Version_func(byte *str)
cmx_reentrant {

    DEC_ASCII(Version,4,1);
    strcpy((char *)str,(char *)AtoD);
    return ((word16)strlen((char *)AtoD));
}
word16 Code_func(byte *str)
cmx_reentrant {
```

```
    DEC_ASCII(Code,3,1);
    strcpy((char *)str,(char *)AtoD);
    return ((word16)strlen((char *)AtoD));
}

#endif        /* #if SERVER_SIDE_INCLUDES */
#endif    /* (EXAMPLE_TYPE == HTTP_EXAMPLE) */
```

7.2.1.2. Callback.c

```
/*************************************************************
Copyright (c) CMX Systems, Inc. 2001. All rights reserved
*************************************************************/

/*************************************************************
* CMX recommends making a copy of this file before changing *
* any of the routines below.                                *
*************************************************************/

#include "micronet.h"

/*************************************************************
   Change the ip_src_addr below to relect the IP address of
   your target. If the IP address of the destination is known
   replace ip_dest_addr with that address. If you are dialing
   into an ISP the source and destination addresses initially
   specified do not matter as they will be negotiated.
*************************************************************/

#if ETHERNET
/* #if (HTTP || FTP || SMTP) */
byte ip_dest_addr[IP_ADDR_LEN] = {216,233,5,32};
byte ip_src_addr[IP_ADDR_LEN]  = {216,233,5,26};
/* #else */
/* byte ip_dest_addr[IP_ADDR_LEN] = {209,67,233,68}; */
/* byte ip_src_addr[IP_ADDR_LEN]  = {209,67,233,67}; */
/* #endif */      /* HTTP || FTP */
#else
byte ip_dest_addr[IP_ADDR_LEN] = {192,6,94,5};
byte ip_src_addr[IP_ADDR_LEN]  = {192,6,94,2};
#endif      /* Ethernet */

#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] = { 0x00,0x00,0x00,0x12,0x34,0x56 };

/*************************************************************
   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 };
#endif      /* ETHERNET */
```

```
/* ---------------------------------------------------------------------- */
/* return number of bytes to send */
/* Called from mn_tcp_send and mn_udp_send */
word16 mn_app_get_send_size(PSOCKET_INFO socket_ptr)
cmx_reentrant {
  word16 bytes_to_send;

  bytes_to_send = (socket_ptr->send_ptr == PTR_NULL ? 0: socket_ptr->send_len);
#if (HTTP || FTP || SMTP)
  if ( (socket_ptr->socket_type & (HTTP_TYPE|SMTP_TYPE)) || \
       socket_ptr->src_port == FTP_DATA_PORT )
    {
    bytes_to_send = ((bytes_to_send > TCP_WINDOW) ? TCP_WINDOW : bytes_to_send);
    }
#endif
  return (bytes_to_send);
}

/* initialization before receiving a packet */
/* Called from mn_tcp_recv and mn_udp_recv */
/* num = number of bytes to be received */
void mn_app_init_recv(word16 num,PSOCKET_INFO socket_ptr)
cmx_reentrant {
#if (!(FTP && FTP_SERVER))
  num = num;
#endif     /* (!(FTP && FTP_SERVER)) */
  socket_ptr->recv_len = 0;

#if HTTP
  if (socket_ptr->socket_type & HTTP_TYPE)
    mn_http_init_recv();
  else
#endif
#if (FTP && FTP_SERVER)
  if (socket_ptr->socket_type & FTP_TYPE)
    mn_ftp_server_init_recv(num,socket_ptr);
  else
#endif
    {
    /* put your code here */
    ;
    }
}

/* process received byte. Called from mn_tcp_recv and mn_udp_recv */
void mn_app_recv_byte(byte c,PSOCKET_INFO socket_ptr)
cmx_reentrant {
#if HTTP
  if (socket_ptr->socket_type & HTTP_TYPE)
    mn_http_recv_byte(c);
  else
#endif
#if (FTP && FTP_SERVER)
  if (socket_ptr->src_port == FTP_CONTROL_PORT)
    mn_ftp_server_recv_byte(c);
  else
#endif
    {
    if (socket_ptr->recv_ptr != PTR_NULL)
      {
      /* copy the data into a buffer where we will deal with it later */
      if ((socket_ptr->recv_ptr+socket_ptr->recv_len) <= socket_ptr->recv_end)
```

```
          {
          *(socket_ptr->recv_ptr+socket_ptr->recv_len) = c;
          socket_ptr->recv_len++;
          }
        }
      }
}

/* Process received packet. Called from mn_tcp_recv and mn_udp_recv. */
void mn_app_process_packet(PSOCKET_INFO socket_ptr)
cmx_reentrant {
#if HTTP
  if (socket_ptr->socket_type & HTTP_TYPE)
    mn_http_process_packet(socket_ptr);
  else
#endif
#if (FTP && FTP_SERVER)
  if (socket_ptr->src_port == FTP_CONTROL_PORT)
    mn_ftp_server_process_packet(socket_ptr);
  else if (socket_ptr->src_port == FTP_DATA_PORT)
    mn_ftp_server_process_data(socket_ptr);
  else
#endif
    {
    /* remove the next line and put your code here */
    socket_ptr = socket_ptr;
    ;
    }
}

#if TCP
/*
  Do what needs to be done after successfully sending a packet.
  Called from mn_tcp_recv().
*/
void mn_app_send_complete(word16 data_len,PSOCKET_INFO socket_ptr)
cmx_reentrant {
  if (socket_ptr->send_len > 0)
    socket_ptr->send_len -= data_len;   /* subtract bytes already sent */

  /* move pointer to next spot to start sending if anything left to send,
    or set it to PTR_NULL otherwise.
  */
  if (socket_ptr->send_len > 0)
    socket_ptr->send_ptr += data_len;
  else
    socket_ptr->send_ptr = PTR_NULL;
}

#endif      /* #if TCP */

/* called from mn_recv */
CHAR mn_app_recv_idle(void)
cmx_reentrant {
  /* put your code here.

    return NEED_TO_EXIT if mn_recv must be exited immediately.
  */
  return (0);
}

#if HTTP
CHAR mn_app_http_server_idle(void)
```

```
cmx_reentrant {
  /* put your code here.
     return NEED_TO_SEND if data must be sent immediately.
  */
  return (0);
}
#endif     /* HTTP */

#if (FTP && FTP_SERVER)
CHAR mn_app_ftp_server_idle(void)
cmx_reentrant {
  /* put your code here.
     return NEED_TO_SEND if data must be sent immediately.
  */
  return (0);
}
#endif     /* #if (FTP && FTP_SERVER) */
```

7.2.1.3. Nec_w.c

```
/*=========================================================================
** PROJECT      = API for CMX Implementation
** MODULE       = NEC_hw.c
** SHORT DESC.  = Hardware setup for CMX stack
** DEVICE       = uPD78F0066(specify change in project options to choose different processor)
** VERSION      = 1.1
** DATE         = 04.01.2002
** LAST CHANGE  = -
** =========================================================================
** Description:  API Implementation
**              The 78K - Test It! demo board implements CMX Internet connectivity stack
**              8-bit timer required stack tick, UART required for connection
**              This file can be modified to change hardware requirements for
**              CMX stack.
**
** =========================================================================
** Environment:  Device:        uPD78F0066
**              Assembler:      A78000        Version     V3.33A
**              C-Compiler:     ICC78000      Version     V3.33A
**              Linker:         XLINK         Version     V4.52J
** =========================================================================
** By:          NEC Electronics (UK) Ltd
**             Cygnus House
**             Sunrise Parkway
**             Linford Wood
**             Milton Keynes UK MK14 6NP
** =========================================================================
Changes:     Incorporate V2.15f into API. Update Interrupt buffer handling
** =========================================================================
*/

#include "micronet.h"

volatile word16 timer_tick;                     /* Restart timer */

#define KEEP_ERROR_COUNT            0                   /* set to 1 to count errors */
#if (KEEP_ERROR_COUNT)
volatile word16 uart_errors;
#endif

extern void vTimer1(void);
```

```
/* ------------------------------------------------------------------------ */
#if (PPP || SLIP)
void mn_uart_init(void)
cmx_reentrant
        {
        DISABLE_INTERRUPTS;
        init_io_buffs();                        /* do not remove this function call */
        /* put uart init code here */
#if   (defined(D78076))
        PM70 = 1;               //Set RXD to input mode
        PM71 = 0;               //Set TXD to output mode
        P7.1 = 1;
#endif
#if   (defined(D78F0034)|| defined(D78F0078))
        PM23 = 1;               //Set RXD to input mode
        PM24 = 0;               //Set TXD to output mode
#endif
#if   (defined(D78F0066))
        PM73 = 1;               //Set RXD to input mode
        PM72 = 0;               //Set TXD to output mode
#endif

#if   (defined(D78F0828))
      PM62 = 1;               //Set RXD to input mode
      PM63 = 0;             //Set TXD to output mode
#endif

#if ((defined(D78F0034)) || (defined(D78F0066)) || (defined(D78F0078)) || (defined(D78F0828)))
        ASIM0 = 0x08;                   /* disable uart No Parity 8bit 1stop*/

        BRGC0 = Baud_38400_4_194304MHz;

      SRIF0  = FALSE;             // Clear Interrupt request reception complete
      STIF0  = FALSE;             // Clear Interrupt request transmision complete
      SRMK0  = FALSE;               // Enable Interrupt reception complete
      STMK0  = FALSE;               // Enable Interrupt transmision complete
        ASIM0 = 0xC8;                 /* Enable UART receive and transmit */
#endif
#if (defined(D78076))
      ASIM = 0x09;                  /* disable uart No Parity 8bit 1stop*/

      BRGC = Baud_38400_5MHz;           /* 38400 @ 5Mhz */

      SRIF  = FALSE;              // Clear Interrupt request reception complete
      STIF  =  FALSE;             // Clear Interrupt request transmision complete
      SRMK  = FALSE;                // Enable Interrupt reception complete
      STMK  =  FALSE;               // Enable Interrupt transmision complete
        ASIM = 0xC9;                  /* Enable UART receive and transmit */
#endif

        ENABLE_INTERRUPTS;
        }
#endif          /* (PPP || SLIP) */

/* put uart isr(s) here */
/* place service routine for receive buffer here */
#if ((defined(D78F0034)) || (defined(D78F0066)) || (defined(D78F0078)) || (defined(D78F0828)))
interrupt [INTSR0_vect] using[3] void vUARTReceive(void)
{
 //     P5.3 = 0;
/*        if (recv_count <= RECV_BUFF_SIZE) */
        if (recv_count < (RECV_BUFF_SIZE-1))          //Fix for buffer management problems
                {
```

```
                        RTS = 0;
                        *recv_in_ptr = RXB0;
                        ++recv_count;
                        ++recv_in_ptr;
                        if (recv_in_ptr > &recv_buff[RECV_BUFF_SIZE-1])
                                recv_in_ptr = &recv_buff[0];
                    }
        else
        {
           RTS = 1;
        }
//          P5.3 = 1;
}
#endif

#if (defined(D78076))
interrupt [INTSR_vect] using[3] void vUARTReceive(void)
{

/*          if (recv_count <= RECV_BUFF_SIZE) */
            if (recv_count < (RECV_BUFF_SIZE-1))            //Fix for buffer management problems
                    {
                        RTS = 0;
                        *recv_in_ptr = RXB;
                        ++recv_count;
                        if (recv_in_ptr > &recv_buff[RECV_BUFF_SIZE-1])
                                recv_in_ptr = &recv_buff[0];
                    }
        else
        {
           RTS = 1;
            }
}
#endif
#if ((defined(D78F0034)) || (defined(D78F0066)) || (defined(D78F0078)) || (defined(D78F0828)))
interrupt [INTSER0_vect] using[3] void eri0(void)
{

//byte temp;
#if (KEEP_ERROR_COUNT)
            uart_errors++;
#endif

            if ( ASIS0 & 0x01 )         ;                               /* overrun error */
//                      temp = RXB0                 /* if errors must read RXB to clear rubbish data */
            else if ( ASIS0 & 0x04 );           /* parity error */

            else if ( ASIS0 & 0x02 );           /* framing error */
            P5 = 0;                             /* Visual Test for errors */
}
#endif

/* place service routine for transmit buffer here */
#if ((defined(D78F0034)) || (defined(D78F0066)) || (defined(D78F0078)) || (defined(D78F0828)))
interrupt [INTST0_vect] using[2] void vUARTTransmit(void)
            {
//        P5.7 = 0;
            if (send_out_ptr != send_in_ptr)
                    {
                      TXS0 = *send_out_ptr;

                ++send_out_ptr;
                    }
```

```
                    else
                    {
                      xmit_busy = 0;
                      STMK0 = 1;
//                P5.7 = 1;
                    }
//      P5.7 = 1;
          }                   /* (PPP || SLIP) */
#endif
#if (defined(D78076))
interrupt [INTST_vect] using[2] void vUARTTransmit(void)
          {
          if (send_out_ptr != send_in_ptr)
                    {
                      TXS = *send_out_ptr;

              ++send_out_ptr;
            }
                else
                    {
                       xmit_busy = 0;
                    }
          }                   /* (PPP || SLIP) */
#endif


/* put timer isr here */
#if ((defined(D78F0034)) || (defined(D78F0066)) || (defined(D78F0078)) || (defined(D78F0828)))
interrupt [INTTM50_vect] using[1] void vTimer(void)
#endif
#if (defined(D78076))
interrupt [INTTM1_vect] using[1] void vTimer(void)
#endif
{
      vTimer1();                  //Added for Dice implementation
          ++timer_tick;
#if DHCP
                    mn_dhcp_update_timer();
#endif                                  /* DHCP */
#if (ARP && ARP_TIMEOUT)
                    mn_arp_update_timer();
#endif              /* (ARP && ARP_TIMEOUT) */

}

void mn_timer_init(void)
cmx_reentrant
          {
          DISABLE_INTERRUPTS;
          /* put timer init code here Must achieve 10ms count for operation */
#if (defined(D78076))
      TMC1    = 0x00;               // Stop timer operation
      TCL1    = 0xFD;               // Set timer clock : fx/2^9 (5MHz -> 9800 Hz)
      CR10    = 0x31;               // Timer1 compare value ->  10.00ms intervall
      TOC1    = 0x03;               // Set to output mode
      PM31    = 0;                 //Set to output Mode
      TMIF1   = FALSE;                // Clear interrupt request Bit Timer1
      TMMK1   = FALSE;                  // Enable Timer1 Interrupt
      TMC1    = 0x01;               // output, no cascade mode
#endif

#if (defined(D78F0828))
      TMC50   = 0x00;                 // Stop timer operation
```

```
      TCL50   = 0x05;                // Set timer clock :   fx/2^8 (4.194304 MHz -> 16384 Hz)
      CR50    = 0x52;                // Timer50 compare value ->  10.01ms intervall
      TMC50   = 0x03;                 // output, no cascade mode
#endif

#if ((defined(D78F0034)) || (defined(D78F0066)) || (defined(D78F0078)))
      TMC50   = 0x00;                 // Stop timer operation
      TCL50   = 0x06;                // Set timer clock :   fx/2^8 (4.194304 MHz -> 16384 Hz)
      CR50    = 0x52;                // Timer50 compare value ->  10.01ms intervall
      TMC50   = 0x03;                 // output, no cascade mode
#endif

#if ((defined(D78F0034))|| (defined(D78F0078)))
      PM72 = 0;                      //Set to output Mode
      TMIF50  = FALSE;                // Clear interrupt request Bit Timer50
      TMMK50  = FALSE;                 // Enable Timer50 Interrupt
      TMC50.7 = 1;
#endif

#if (defined(D78F0066))
      PM22 = 0;                      //Set to output Mode
      TMIF50  = FALSE;                // Clear interrupt request Bit Timer50
      TMMK50  = FALSE;                 // Enable Timer50 Interrupt
      TMC50.7 = 1;
#endif

#if (defined(D78F0828))
      PM34 = 0;                      //Set to output Mode
      TMIF50  = FALSE;                // Clear interrupt request Bit Timer50
      TMMK50  = FALSE;                 // Enable Timer50 Interrupt
      TMC50.7 = 1;
#endif

        timer_tick = 0;
        ENABLE_INTERRUPTS;
}
```

7.2.1.4. Init.c

```
/*===========================================================================
** PROJECT      = 78K0 Demonstration Board
** MODULE       = init.c
** SHORT DESC.  = -
** DEVICE       = uPD78F0066
** VERSION      = 1.0
** DATE         = 27.11.2001
** LAST CHANGE  = -
** ===========================================================================
** Description:  Initialization of Registers and Bit Variables
** ===========================================================================
** Environment:  Device:        uPD78F0066
**               Assembler:     A78000        Version     V3.33A
**               C-Compiler:    ICC78000      Version     V3.33A
**               Linker:        XLINK         Version     V4.52J
** ===========================================================================
** By:          NEC Electronics (UK) Ltd
**              Cygnus House
**              Sunrise Parkway
**              Linford Wood
**              Milton Keynes UK MK14 6NP
** ===========================================================================
Changes:
```

```
** ============================================================================
*/

//----------------------------------------------------------------------------
// Include files
//----------------------------------------------------------------------------
#include <in78000.h>
#if (defined(D78F0034))
  #include <df0034.h>                    /* change to appropriate file */
#endif
#if (defined(D78F0066))
  #include <df0066.h>                    /* change to appropriate file */
#endif
#if (defined(D78F0078))
  #include <df0078.h>                    /* change to appropriate file */
#endif
#if (defined(D78F0828))
  #include <df0828.h>                  /* change to appropriate file */
#endif
#if (defined(D78076))
  #include <d076.h>                  /* change to appropriate file */
#endif

#include "micronet.h"

//----------------------------------------------------------------------------
// Global variables
//----------------------------------------------------------------------------
extern bit          bTimer50Flag;     // Status Flag Timer50
extern bit          bTimer51Flag;     // Status Flag Timer51
extern bit          bKey1Flag;        // Status Flag Key1
extern bit          bKey2Flag;        // Status Flag Key2
extern unsigned int Compiler_Version_Number;

/* ============================================================================
**  Module name: vHardwareInit
**
**  Description:
**          This module is to initialize some peripheral hardware.
**
**  Operation:
**          Sets the clock generator, the port modes and output latches
**          and initializes the interrupts.
** ============================================================================
*/
void vHardwareInit(void)              // Hardware inizialization
{
#if (defined(D78F0066))
// clock generator setting
//------------------------
   PCC  = 0x00;                    // Use high speed mode
   OSTS = 0x02;                     // 2^15/fx ms wait after STOP release by interrupt
   IMS  = 0xCC;                    // Select 1024 Byte RAM and 48k Byte ROM
   IXS  = 0x04;                   // Select 4096 Byte RAM

// port setting
//-------------
   P0    = 0x00;                  // Set output latch to 0
   P2    = 0x00;                  // Set output latch to 0
   P3    = 0x00;                  // Set output latch to 0
   P4    = 0x00;                  // Set output latch to 0
   P5    = 0xff;                 // Set output latch to 0xff
   P6    = 0x00;                  // Set output latch to 0
```

```
    P7     = 0x01;                  // Set output latch to 0x01 (enable RS232 driver)
    P8     = 0x00;                  // Set output latch to 0
    P8     = 0x00;                  // Set output latch to 0
    PU7    = 0x08;                   // Enable internal pull-up resistor P7.3
    PM0    = 0xFF;                   // Port 0 =  input (8-bits)
    PM2    = 0xFF;                   // Port 2 =  input (8-bits)
    PM3    = 0xFF;                   // Port 3 =  input (8-bits)
    PM4    = 0x00;                   // Port 4 =  input (8-bits)
    PM5    = 0x00;                   // Port 5 = output (8-bits)
    PM6    = 0xFF;                   // Port 6 =  input (4-bits)
    PM7    = 0x88;                   // Port 7.7,7.3 input, other output
    PM8    = 0xE0;                   // Port 8 = output (5 bits)
    PM9    = 0xFF;                   // Port 9 =  input (3-bits)

// interrupt setting
//------------------

    IF0L   = 0x00;                  // Reset all interrupt request bits
    IF0H   = 0x00;
    IF1L   = 0x00;
    MK0L   = 0xFF;                   // Disable all interrupts
    MK0H   = 0xFF;
    MK1L   = 0xFF;
    PR0L   = 0xFF;                   // Set all interrupts to low priority
    PR0H   = 0xFF;
    PR1L   = 0xFF;

// Analog Converter setting
// -------------------------
                              // Stop AD-Converter, start conversion by SW
    ADM0   = 0x00;                  // conversion time 144/fx (-> 17.1µs)
    ADS0   = 0x00;                  // Default selection: channel 0
    ADIF0  = FALSE;                   // Clear Interrupt request bit ADC
    ADMK0  = FALSE;                     // Enable ADC Interrupt

// ext. Interrupt setting
// -------------------
    EGP    = 0x0C;                  // Enable ext.interrupt rising edge
    EGN    = 0x00;                  // P2, P3
    PMK3   = FALSE;                    // Enable Port P0.3 interrupt (Mask Flag Register)
    PMK2   = TRUE;                     // Disable Port P0.2 interrupt (Mask Flag Register)

// 8 Bit Timer setting    Timer 50 setup in NEC_HW_v2_15f for Micronet Stack implementation
// -------------------
    TMC51  = 0x00;                   // Stop timer operation
    TCL51  = 0x07;                  // Set timer clock :   fx/2^11 (4.914304MHz -> 2048 Hz)
    CR51   = 0x66;                  // Timer51 compare value =>  50ms intervall
    TMIF51 = FALSE;                    // Clear interrupt request Bit Timer51
    TMMK51 = FALSE;                      // Enable Timer51 Interrupt
    TMC51  = 0x00;                   // Timer 51 setting
                              // no output, single mode
                              // clear and start by matching TM51 and CR51

// 16 Bit Timer setting
// --------------------

    TMC0   = 0x00;                  // Stop and Clear Timer 0
    CRC0   = 0x00;                  // Register CR00, CR01 operate as compare register
    CR00   = 0x028f;                 // Compare values for 10ms Intervall
    TOC0   = 0x00;                  // Disable Timer output
    PRM0   = 0x02;                  // Clock selection: fx/2^6 (4.914304MHz -> 65537 Hz)
    TMIF00 = FALSE;                    // Clear interrupt request Bit Timer0
    TMMK00 = FALSE;                      // Enable Timer0 Interrupt
```

```
   TMC0   = 0x02;                  // Clear and start on match between TM0 and CR00
   TMC0 = 0x0C;                    // Start Timer 0


// Compiler Version Number setting
// ---------------------

   Compiler_Version_Number = __VER__;            //Must be left in code so Micronet will work with current
compiler version
#endif

#if (defined(D78076))
// clock generator setting
//------------------------
   PCC  = 0x00;                    // Use high speed mode
   OSTS = 0x02;                    // 2^15/fx ms wait after STOP release by interrupt
   OSMS = 0x01;                    // Scalar not used
   IMS  = 0xCC;                    // Select 1024 Byte RAM and 48k Byte ROM
   IXS  = 0x0A;                    // Select 1024 Byte RAM
#endif
}

/* ============================================================================
**  Module name: vSoftwareInit
**
**  Description:
**            This module is to initialize the bit variables
**
**  Operation:
**            -
** ============================================================================
*/
void vSoftwareInit(void)
{
   bTimer50Flag = FALSE;           // Reset status flag Timer50
   bTimer51Flag = FALSE;           // Reset status flag Timer51
   bKey1Flag    = FALSE;           // Reset status flag Key1
}
```

7.2.1.5. Conversion.c

```
/*============================================================================
** PROJECT      = 78K0 Demonstration Board
** MODULE       = conversion.c
** SHORT DESC.  = -
** DEVICE       = uPD78F0066
** VERSION      = 1.1
** DATE         = 05.02.2002
** LAST CHANGE  = -
** ============================================================================
** Description:  Serial conversion routines
**
** ============================================================================
** Environment:  Device:        uPD78F0066
**               Assembler:     A78000        Version     V3.33A
**               C-Compiler:    ICC78000      Version     V3.33A
**               Linker:        XLINK         Version     V4.52J
** ============================================================================
** By:           NEC Electronics (UK) Ltd
**               Cygnus House
**               Sunrise Parkway
**               Linford Wood
```

```
**              Milton Keynes UK MK14 6NP
** ============================================================================
Changes:
** ============================================================================
*/


//----------------------------------------------------------------------------
// Include files
//----------------------------------------------------------------------------
#include <in78000.h>
#include <string.h>
#if (defined(D78F0066))
    #include "df0066.h"
#endif
#include "micronet.h"

//----------------------------------------------------------------------------
// Global variables
//----------------------------------------------------------------------------
bit bTransmit;
bit bADCFlag;
byte AtoD[4];
byte AtoD_buff[8];

//----------------------------------------------------------------------------
// Function prototyps
//----------------------------------------------------------------------------

/*****************************************************************************=*****
* Conversion from ASCII to DEC
*
*****************************************************************************=*****/

byte ASCII_DEC(byte a)
{
byte DEC;

   if ((a < ':')&& ( a > '/'))
      DEC = (a -'0');
   else
      DEC = 0xAA;
  return(DEC);
}

/*****************************************************************************=*****
* Conversion from DEC to ASCII byte only
*  Example ASCII = 12     output to HTML page = 1.2
a = 12, b = 3, c = 1
*  AtoD          0     1     2     3
       DEC      1     .     2
       ASCII    49    46    50    13
*  Example ASCII = 215     output to HTML page = 2.15
a = 215, b = 4, c = 1
  AtoD          0     1     2     3     4
       DEC      2     .     1     5
       ASCII    50    46    49    53    13
*****************************************************************************=*****/

void DEC_ASCII(byte a, byte b, byte c)
{
unsigned char ASCII;
unsigned char temp;
```

```
  ASCII = a;
   AtoD[b] = '\0';                //Set End of AtoD buffer to Carriage return for strcopy command AtoD[3] = 13
   if (c >= 1)
     AtoD[c] = '.';                // Insert decimal point  AtoD[1] = '.'

   if (ASCII>=100)
   {
     temp = ASCII%100;
     ASCII /= 100;
     if(c >= 1)
       AtoD[4-b] = '0' + ASCII;          //ASCII = 2,     AtoD[0] = '2'
       ASCII = temp;
   }
   if (ASCII >= 10)                //
   {
     AtoD[b-1] = '0' + ASCII%10;       //ASCII = 15,    AtoD[2] = 'ASCII MOD 10' = '5'
     ASCII /= 10;
     if((c >= 1)&& (b == 3))
       AtoD[b-3] = '0' + ASCII;          //ASCII = 1,     AtoD[0] =
     if((c >= 1)&& (b == 4))
       AtoD[b-2] = '0' + ASCII;          //ASCII = 1,     AtoD[0] =
   }
   else
   {
     AtoD[1-b] = '0' + ASCII;          //If ASCII = 9,  AtoD[2] = '9'
     if (c >= 1)
       AtoD[c-1] = '0';                //AtoD[0] = '0'
   }

}
```

7.2.1.6. AtoD.c

```
/*===========================================================================
** PROJECT      = 78K0 Demonstration Board
** MODULE       = AtoD.c
** SHORT DESC.  = -
** DEVICE       = uPD78F0066
** VERSION      = 1.0
** DATE         = 04.02.2002
** LAST CHANGE  = -
** ===========================================================================
** Description:  A to D demonstration of Test-It! board using CMX Micronet
**               The demo boards logs the analog ports in defined time
**               time intervals.
**
* Environment:  Device:         uPD78F0066
**               Assembler:      A78000         Version     V3.33A
**               C-Compiler:     ICC78000       Version     V3.33A
**               Linker:         XLINK          Version     V4.52J
** ===========================================================================
** By:           NEC Electronics (UK) Ltd
**               Cygnus House
**               Sunrise Parkway
**               Linford Wood
**               Milton Keynes UK MK14 6NP
** ===========================================================================
Changes:
** ===========================================================================
*/
```

```
//-----------------------------------------------------------------------------
// Include files
//-----------------------------------------------------------------------------
#include <in78000.h>
#if (defined(D78F0066))
  #include "df0066.h"
#endif
#include "micronet.h"

#if ((defined(D78F0034)) || (defined(D78F0066)) || (defined(D78F0078)) || (defined(D78F0828)))
//-----------------------------------------------------------------------------
// Function prototyps
//-----------------------------------------------------------------------------
unsigned char       ucGetAD       (unsigned char);  // Read ADC result of one channel

//-----------------------------------------------------------------------------
// Global variables
//-----------------------------------------------------------------------------
extern bit          bADCFlag;         // Status flag ADC
extern byte AtoD_buff[8];
//extern bit          bTransmit;        // Status flag Transmit complete

/* ============================================================================
**  Module name: AtoD_Values
**
**  Description:
**              This module gets the A to D values and stores them back in AtoD_buff
**
**  Operation:
**              -
** ============================================================================
*/
void AtoD_Values(void)
{
byte i;
word16 temp;
word16 y;


        P8.0 = TRUE;                // Switch on potentiomer supply
        P7.4 = TRUE;                // Switch on potential divider supply
        P7.5 = TRUE;                // Switch on reference diode supply
        for (i = 0;i < ucAChannel;i++)
        {
          AtoD_buff[i] = ucGetAD(i);  // Store result of each analog port
        }
        P8.0 = FALSE;                // Switch off potentiomer supply
        P7.4 = FALSE;                // Switch off potential divider supply
        P7.5 = FALSE;                // Switch off reference diode supply

        y = ADCscalar/AtoD_buff[0]/10;    // Work out y scalar based on ADC Channel 0 = voltage reference
1.25V

        for (i = 0;i < ucAChannel;i++) // Calculate all ADC values against Scalar
        {
          if (i == 3)
          {
             temp = AtoD_buff[i] * y;
             temp /= 127;
             AtoD_buff[i] = temp;
          }
          else
          {
```

```
            temp = AtoD_buff[i] * y;
            temp /= 255;
            AtoD_buff[i] = temp;
          }
        }
}

/* ============================================================================
**  Module name: ucGetAD
**
**  Description:
**            This module returns the AD-result of the selected channel
**            channel range 0 ... 7
**  Operation:
**            -
** ============================================================================
*/
unsigned char ucGetAD(unsigned char ucChannel)
{
   ADM0.7   = TRUE;                // Start AD-Conversion
   ADS0     = ucChannel;           // Channel selection
   while (bADCFlag == FALSE){
   }
   bADCFlag = FALSE;               // Reset status flag ADC
   while (bADCFlag == FALSE){
   }
   bADCFlag = FALSE;               // Reset status flag ADC
   ADM0.7   = FALSE;               // Stop AD-Conversion
   return ADCR0;
}

/* ============================================================================
**  Module name: vADCComplete
**
**  Description:
**            This module is the Interrupt service routine for the
**            A/D - converter
**
**  Operation:
**            -
** ============================================================================
*/
interrupt [INTAD0_vect] void vADCComplete(void)
{
    bADCFlag = TRUE;               // Set flag
}

#endif
```

7.2.1.7. Dice.c

```
/*============================================================================
** PROJECT      = 78K0 Demonstration Board
** MODULE       = dice.c
** SHORT DESC.  = -
** DEVICE       = uPD78F0066
** VERSION      = 1.0
** DATE         = 02.01.2000
** LAST CHANGE  = -
** ============================================================================
** Description:  Electronic Dice
**            After initialization all LEDs flash twice
```

```
**              After pressing Key1, one to six LEDs get switched on.
**              Then the program waits for the next start.
**
** ============================================================================
** Environment:  Device:         uPD78F0066
**              Assembler:      A78000         Version 3.30A
**              C-Compiler:     ICC78000       Version 3.30A
**              Linker:         XLINK          Version 4.51O
**              Simulator:      SM78K0         Version 2.10
** ============================================================================
** By:          NEC Electronics (Europe) GmbH
**              Oberrather Strasse 4
**              D-40472 Duesseldorf
**
** ============================================================================
Changes:
** ============================================================================
*/

//----------------------------------------------------------------------------
// Defines
//----------------------------------------------------------------------------

//----------------------------------------------------------------------------
// Include files
//----------------------------------------------------------------------------
#include <in78000.h>
#include "DF0066.h"
#include "micronet.h"

//----------------------------------------------------------------------------
// Function prototyps
//----------------------------------------------------------------------------
      void          vFlashLED      (unsigned char);// Flash all LEDs n times
      void          vWait5         (unsigned char);// n *  5ms delay
      void          dice           (void);
      unsigned char ucRandom       (void);         // Generation of random-number

//----------------------------------------------------------------------------
// Global variables
//----------------------------------------------------------------------------
bit          bTimer50Flag;                  // Status Flag Timer50
bit          bTimer51Flag;                  // Status Flag Timer51
bit          bKey1Flag;                     // Status Flag Key1
saddr unsigned char ucThrow;                      // dice value
saddr unsigned char ucLEDs=0xff;                  // output value LEDs

/* ============================================================================
**  Module name: ucRandom
**
**  Description:
**              This module generates a pseudo-random number.
**
**  Operation:
**              -
** ============================================================================
*/
unsigned char ucRandom(void)
{
   return (((TM50)%6)+1);             // pseudo-random number between 1 ... 6
}
```

```
/* ============================================================================
**  Module name: vFlashLED
**
**  Description:
**              This module flashes all LEDs n times.
**
**  Operation:
**              -
** ============================================================================
*/
void vFlashLED(unsigned char Number)
{
   while(Number>0) {
     ucLEDs=0x00;
     vWait5(10);                // Delay of 500 ms
     ucLEDs=0xff;
     vWait5(10);                // Delay of 500 ms
     Number--;
   }
   return;
}


/* ============================================================================
**  Module name: vWait5
**
**  Description:
**              This module delays the program for n * 5 ms.
**
**  Operation:
**              -
** ============================================================================
*/
void vWait5(unsigned char Number)
{
   bTimer51Flag=0;              // Reset Timer51 Flag
   TMC51.7 = 1;                 // Start Timer51
   while(Number>0) {
      while(bTimer51Flag==0){
      }
      bTimer51Flag=0;           // Reset status flag Timer51
      Number--;
   }
   TMC51.7 = 0;                 // Stop Timer51
   return;
}


/* ============================================================================
**  dice main function
** ============================================================================
*/
void dice( void )
{
   unsigned char i;
   vFlashLED(2);                // Signal: end of initialization
     if(bKey1Flag==1){
        bKey1Flag=0;            // Reset status bit key1
        ucThrow=ucRandom();     // Get random number
        ucLEDs=0xFE;
        for (i=1;i<ucThrow;i++)
          ucLEDs=(ucLEDs<<1);   // Output of new value
     }
```

```
}
```

7.2.1.8. Interrupt.c

```
/*============================================================================
** PROJECT      = 78K0 Demonstration Board (Electronic Dice)
** MODULE       = interrupt.c
** SHORT DESC.  = -
** DEVICE       = uPD78F0066
** VERSION      = 1.1
** DATE         = 05.02.2002
** LAST CHANGE  = -
** ============================================================================
** Description:  Interrupt service routines for Electronic Dice
**
** ============================================================================
** Environment:  Device:        uPD78F0066
**               Assembler:     A78000        Version    V3.33A
**               C-Compiler:    ICC78000      Version    V3.33A
**               Linker:        XLINK         Version    V4.52J
** ============================================================================
** By:          NEC Electronics (UK) Ltd
**              Cygnus House
**              Sunrise Parkway
**              Linford Wood
**              Milton Keynes UK MK14 6NP
** ============================================================================
Changes:     Incorporate V2.15f into API. Update Interrupt buffer handling
** ============================================================================
*/

//----------------------------------------------------------------------------
// Defines
//----------------------------------------------------------------------------

//----------------------------------------------------------------------------
// Include files
//----------------------------------------------------------------------------
#include <in78000.h>
#include "df0066.h"
#include "micronet.h"

//----------------------------------------------------------------------------
// Global variables
//----------------------------------------------------------------------------
extern bit          bTimer50Flag;     // Status Flag Timer50
extern bit          bTimer51Flag;     // Status Flag Timer51
extern bit          bKey1Flag;        // Status Flag Key1
extern bit          bKey2Flag;        // Status Flag Key2

extern saddr unsigned char ucLEDs;
static saddr unsigned char ucOldLEDs;

void vTimer1(void);
/* ============================================================================
**  Module name: vTimer0
**
**  Description:
**              This module is the Interrupt service routine for Timer0
**
**  Operation:
**
```

```
** ============================================================================
*/
interrupt [INTTM00_vect] void vTimer0(void)
{
   static saddr unsigned char ucFlag;
   static saddr unsigned char ucOldLEDs;
   if (ucOldLEDs==ucLEDs){
      if (ucFlag==0x01){
        ucFlag=0x00;
        LED=0xff;                    // all LEDs off
      }
      else{
        ucFlag=0x01;
        LED=ucLEDs;                  // Output LED value
      }
   }
   else{
     ucOldLEDs=ucLEDs;                    // save current LED value
     LED=ucLEDs;                  // Output LED value
     ucFlag=0x01;
   }
}


/* ============================================================================
**  Module name: vTimer1
**
**  Description:
**            This module is the Interrupt service routine for Timer50 added procedure to NEC_HW_v2_15f
**
**  Operation:
**            -
** ============================================================================
*/
void vTimer1(void)
{
   bTimer50Flag = 1 ;               // Set status flag Timer50
}


/* ============================================================================
**  Module name: vTimer51
**
**  Description:
**            This module is the Interrupt service routine for Timer51
**
**  Operation:
**            -
** ============================================================================
*/
interrupt [INTTM51_vect] void vTimer2(void)
{
   bTimer51Flag = 1 ;               // Set status flag Timer51
}


/* ============================================================================
**  Module name: vKey1
**
**  Description:
**            This module is the Interrupt service routine for
**            the external interrupt P3 (-> Key1)
**
```

```
**  Operation:
**              -
** ============================================================================
*/
interrupt [INTP3_vect] void vKey1(void)
{
   bKey1Flag = 1 ;                  // Set status flag Key1
}
```

Index

ACTIVE_OPEN.. 30, 34, 36, 44, 46, 48, 64, 66, 67
ACTIVE_OPEN;.....44, 64
ANSWER_MODE 38, 42, 44, 61, 63
ANSWER_MODE; 44, 63
Application Layer 9, 11, 15
Application Ports.....12
ARP..... 9, 12, 13, 52, 57, 72, 78
ArrayTotal 19

BODY_BUFFER 19, 21
BOOTP 9, 11, 12, 44, 45, 46, 63, 65

Callback 16, 24, 25, 27, 33, 36, 42, 51, 71
Client.....14

DHCP..... 9, 12, 45, 46, 57, 65, 66, 78
DIAL_MODE 38, 42, 44, 61, 64
DIAL_MODE; 44, 64
DIRECT_CONNECT.....31
DNS 11

EXAMPLE_TYPE30, 42, 43, 45, 46, 47, 49, 50, 51, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71
Exec cgi 26, 34, 40

FTP9, 11, 12, 20, 27, 30, 42, 43, 45, 46, 51, 52, 53, 54, 61, 62, 64, 65, 66, 72, 73, 74
FTP Server 12, 27
FTP_NUM_USERS 20

GET.....20, 33, 41

HTML9, 12, 13, 15, 21, 25, 26, 27, 30, 31, 33, 39, 40, 61, 83
Html2c.exe.....30
HTTP9, 11, 12, 13, 14, 15, 19, 21, 26, 27, 30, 31, 33, 42, 43, 45, 46, 50, 51, 52, 53, 54, 61, 62, 63, 64, 65, 66, 69, 70, 71, 72, 73, 74
HTTP Server.....12, 14, 31
HTTP_BUFFER_LEN.....21
HTTPTotal.....19

ICMP 9, 10, 11, 15, 36
INTSER0.....23, 56, 77
INTSR0 .....23, 56, 76
INTST0.....23, 57, 77

JAVA 13

Link Layer 9, 15, 18, 21

Main8, 10, 11, 12, 28, 30, 40, 44, 45, 61, 63, 64, 88
Memory Map 17
Mn_app_http_server_idle 36, 54, 74
Mn_app_init_recv 33, 37, 52, 73
Mn_app_process_packet.....33, 38, 53, 73
Mn_app_recv_byte 33, 38, 52, 73
Mn_app_recv_idle 38, 54, 74
Mn_gf_del_entry 41
Mn_gf_get_entry 34, 40
Mn_gf_set_entry 31, 41, 45, 64
Mn_http_find_value..... 33, 50, 69, 70
Mn_http_init_recv..... 33, 52, 73
Mn_http_process_packet..... 33, 53, 73
Mn_http_recv_byte 33, 53, 73
Mn_http_server 33, 46, 66
Mn_init 30, 33, 44, 64
Mn_ip_recv 36
Mn_modem_connect 31, 38, 45, 65
Mn_modem_disconnect..... 32, 38, 45, 46, 65, 66
Mn_modem_send_string 38
Mn_modem_wait_reply.....38
Mn_pf_del_entry 41
Mn_pf_get_entry 33, 41
Mn_pf_set_entry 30, 41, 45, 64
Mn_ppp_close..... 31, 36, 46, 66
Mn_ppp_open 36, 45, 65
Mn_ppp_reset 36
Mn_tcp_abort 34, 35
Mn_tcp_open 34
Mn_tcp_recv 34, 52, 53, 73, 74
Mn_tcp_send 34, 52, 72
Mn_tcp_shutdown.....34
Mn_vf_del_entry 39
Mn_vf_get_entry 33, 39
Mn_vf_set_entry 30, 39, 40, 64
MODEM9, 27, 30, 31, 32, 34, 36, 38, 42, 44, 45, 46, 61, 63, 64, 65, 66
MODEM_MODE 30, 42, 44, 61, 63

Network Layer.....9, 15
Null MODEM 31
NULL_MODEM 31
NUM_GET_FUNCS.....20
NUM_POST_FUNCS.....20
NUM_SOCKETS.....20
NUM_VF_PAGES.....20

PAP..... 9, 11, 12, 20, 45, 65
PAP_NUM_USERS 20
PASSIVE_OPEN 30, 34, 36, 44, 46, 48, 64, 66, 67
PASSIVE_OPEN; 44, 64
Physical Layer..... 12, 18
PING 9, 11, 15, 36
POST6, 9, 13, 15, 19, 20, 21, 30, 31, 33, 40, 41, 50, 69, 70
PPP9, 12, 15, 18, 20, 30, 31, 33, 34, 36, 44, 45, 46, 47, 48, 49, 55, 56, 57, 63, 64, 65, 66, 67, 68, 75, 76, 77, 78

RARP 12, 13
RECV_BUFF_SIZE..... 19, 21, 56, 76, 77
recv_in_ptr 56, 76, 77
recv_in_ptr; 56, 76

send_out_ptr 57, 77, 78
send_out_ptr; 57, 77, 78

Server 9, 12, 13, 14, 15, 19, 20, 21, 26, 27, 31, 40
SLIP 9, 12, 18, 36, 55, 56, 57, 75, 76, 77, 78
SMTP9, 11, 12, 14, 27, 42, 43, 46, 49, 50, 51, 52, 61, 62, 63, 66, 68, 69, 72

TCP6, 9, 10, 11, 13, 15, 19, 20, 21, 27, 33, 34, 36, 38, 42, 43, 46, 47, 48, 49, 52, 53, 54, 61, 62, 63, 65, 67, 68, 72, 74
TCP/IP Protocol Stack 9
TCP_WINDOW 19, 21, 52, 72
TFTP 9, 11, 12, 44, 45, 46, 63, 65
TM50 ..... 23, 87

UART ... 8, 9, 15, 18, 23, 44, 54, 55, 56, 64, 75, 76
UDP9, 10, 11, 27, 38, 42, 43, 46, 61, 62, 63, 65, 66
URI_BUFFER_LEN 19, 21

Virtual Files 18, 21

XMIT_BUFF_SIZE 21
XML 13