NetChip
Technology, Inc.

*335 Pioneer Way*
*Mt View, California 94041*
*(650) 526-1490  Fax (650) 526-1494*
*e-mail: sales@netchip.com*
*Internet: www.netchip.com*

# NET2888 Evaluation Board

# User's Manual

for NET2888-EB Rev 4 Evaluation Board

**Doc #: 610-0004-0004**
**Revision:  0.4**
**Date:  10 / 28 / 97**

# Table of Contents




**625 Clyde Avenue, Mountain View, California 94043**
TEL (650) 526-1490 FAX (650) 526-1494
http://www.netchip.com

# PART I
# THE NET2888-EB

This section describes installation of the NET2888-EB kit and the evaluation board hardware.

# 1. Installation of NET2888-EB

Please follow these steps to install and use the NET2888-EB.

## *1.1. System Requirements*

- Microsoft Memphis Beta OS (Build 14xx or later). Memphis is available to peripheral developers through a non-disclosure agreement with Microsoft Corporation.
- Host PC with USB motherboard or USB plug-in card.
- (RECOMMENDED) Null modem serial cable (recommended to monitor serial port 0 on NET2888-EB with host computer).
- (OPTIONAL) WINUTB.EXE, available from Microsoft Corp. on the WDM DDK CD.

## *1.2. Installation Steps*

**Step 1. Jumper Settings**

The NET2888-EB's default jumper settings are shown in the following table. Please confirm the jumper settings before plugging in the board.

***Table 1-1: Jumper Settings***

| Jumper | Factory Setting |
|---|---|
| JP0201 | 1-2 ON |
| JP0202 | 1-2 ON |
| JP0203 | 1-2 ON |
| JP0204 | 1-2 ON |
| JP0205 | 1-2 ON |
| JP0301 | 1-2 ON |
| JP0302 | OPEN |
| JP0303 | OPEN |
| JP0304 | 1-2 ON |
| JP0701 | 1-2 ON |
| JP0801 | OPEN |

**Step 2. Power Up your Host PC**

Turn on the Host PC, and boot up Microsoft Memphis OS.

**Step 3. Connect the USB Cable**

The rectangular connector on the USB cable plugs into the USB port of your Host PC, or downstream port of a USB Hub. The square end plugs into the NET2888-EB. It is impossible to incorrectly connect the USB cable because the connectors are keyed.

The first time you plug in the NET2888-EB evaluation board, the host will ask for a driver. Insert the included floppy disk, and select the "NET2888-EB in Loopback Mode" driver.

# 2. Getting Started

## *2.1. Board Initialization*

When the board is reset (e.g. on plug-in) the LED on the NET2888-EB will flash briefly several times to indicate that the board is operating properly. If the LED does not flash when reset, or the LED turns on and stays on, the board is not functioning (executing from the local PROM) correctly.

## *2.2. Loopback Self-Test*

When the NET2888-EB is plugged into the USB with the default factory jumper settings as shown in Section 1.2, the board will automatically enter loopback mode. In loopback mode, the NET2888 will receive data from the host PC and immediately send it up to the host again.

There are two applications which can test loopback mode of the NET2888-EB: the NetChip Host Monitor (provided on the floppy disk with this board), and Microsoft's WINUTB.EXE (provided on the WDM DDK CD).

**NOTE**: The NET2888-EB must be in loopback mode (see *LocalMon* `mode` command) to successfully complete these tests.

### 2.2.1. Loopback with NetChip's Host Monitor (HostMon)

The Host Monitor program provides the ability to do loopback testing. After the NET2888-EB has been connected to the USB, run the host monitor application (HOSTMON.EXE on the floppy disk).

To test the NET2888 mailbox registers (Endpoints 1 and 2), enter the following at the *HostMon* command prompt:

```
loopmb; r;
```

This test will write random data to the receive mailbox (endpoint 1) and read data from the transmit mailbox (endpoint 2). The received data is compared to the transmitted data, and any error is reported to the console. The 'r' causes the test to repeat until a key is pressed.

To test the NET2888 FIFOs (Endpoints 3 and 4), enter the following at the *HostMon* command prompt:

```
loopfi; r;
```

This test will write random data to the receive FIFO (endpoint 3) and read data from the transmit FIFO (endpoint 4). The received data is compared to the transmitted data, and any error is reported to the console. The 'r' causes the test to repeat until a key is pressed.

`loopfi`, `loopmb`, and other *HostMon* commands are fully described in Section 7.

### 2.2.2. Loopback with Microsoft's WINUTB.EXE

From Microsoft's WDM DDK CD, find the source files related to WINUTB.EXE. You will need to compile this program, following the directions in the DDK.

Once you have compiled WINUTB, run the program. Select the *Open Loop* choice from the application's *File* menu. You will be presented with a list of USB devices currently connected. The one labeled Vendor_ID (0525) and Product_ID (2888) is the NET2888-EB. Select this device, and press OK.

Next, select *Do Loopback* from the *File* menu. From this dialog box, set the output pipe to Pipe 2, and the input pipe to Pipe 3. Select buffer size = 64 or smaller for both pipes, and select 100 iterations. Other options may be left at their default values.

When you press OK from this dialog box, the host PC will transfer data (64 bytes at a time) to the receive FIFO (endpoint 3) on the NET2888, and receive the same data from the transmit FIFO (endpoint 4). The two sets of data are compared, and any errors are placed on the screen.

Similarly, the loopback test may be performed over endpoints 1 and 2 (the NET2888 Receive and Transmit Mailboxes). Set the output pipe to Pipe 0, the input pipe to Pipe 1, and a buffer size of 8 or smaller.

## *2.3. USB Transaction Examples*

The NET2888-EB has the capability of receiving and transmitting USB data packets to a USB Host PC. The *LocalMon* and *HostMon* applications are fully featured monitor programs which run on the NET2888-EB and host PC, respectively. This section describes a few command sequences to give examples of transmitting and receiving data using the *LocalMon* and *HostMon* applications. Follow the steps below.

### 2.3.1. Setting Up

Follow the directions in Section 8.1 to establish a connection to the *LocalMon* through NET2888-EB's serial port 0. On the Host PC, run HostMon.EXE (on the included floppy disk).

### 2.3.2. Transferring 64 bytes from Host PC to NET2888-EB through Receive FIFO

1. At *LocalMon*'s '&' command prompt, type:
   `& mode 0`
   Put the NET2888-EB into the mode where it does not process USB packets automatically (so that we can do it manually from *LocalMon*)
2. At *HostMon*'s '&' command prompt, type:
   `& f obuf l 40 aa 55`
   Fill *obuf* with the repeating pattern "aa 55" of length 64 (hex 40).
   NOTE: The letter after obuf is a lowercase 'L', not a one.
   `& ofi`
   Output 64-byte *obuf* over USB to NET2888 Receive FIFO.
3. At *LocalMon*'s '&' command prompt, type:
   `& rfi`
   Read NET2888 Receive FIFO into *rbuf*.
   `& d rbuf`
   Display memory range *rbuf*.
   `& RfiCount`
   Display contents of variable RfiCount (number of bytes read).

The NET2888-EB has now received 64 bytes of the repeating "aa 55" pattern from the Host PC into the local memory range *rbuf*.

### 2.3.3. Transferring 64 bytes from NET2888-EB to Host PC through Transmit FIFO

1. At *LocalMon*'s '&' command prompt, type:
   `& mode 0`
   Put the NET2888-EB into the mode where it does not process USB packets automatically (so that we can do it manually from *LocalMon*)
   `& f tbuf l 40 aa 55`
   Fill *tbuf* memory buffer with pattern "aa 55" for 64 bytes (hex 40).
   NOTE: The letter after `tbuf` is a lowercase 'L', not a one.
   `& tfi`
   Load *tbuf* into NET2888 Transmit FIFO for transmission to USB host.
   `& TfiCount`
   Display the number of bytes written to the transmit FIFO.

2. At *HostMon*'s '&' command prompt, type:
   `& ifi`
   Read 64 bytes from NET2888 Transmit FIFO, place into *ibuf*.
   `& d ibuf`
   Display memory range *ibuf*.

The Host PC has now received 64 bytes of the repeating "aa 55" pattern from the NET2888-EB, and placed it into the host's memory range, *ibuf*.

# 3. NET2888 Hardware Description

8051 CPU crystal frequency = 11.059 MHz
NET2888 I/O crystal frequency = 48.000 MHz

## *3.1. Jumper Description*

| Jumper | Description |
|---|---|
| JP0201 | **BUSPWR#**<br>1-2 = Bus Powered<br>2-3 = Self Powered |
| JP0202 | **Measure NET2888 Current**<br>1-2 = Normal Operation |
| JP0203 | **Measure NET2888-EB Board Current**<br>1-2 = Normal Operation |
| JP0204 | **NET2888 Test Mode**<br>1-2 = Normal Operation<br>2-3 = Enable NET2888 Test Mode |
| JP0205 | **NET2888 I/O Current**<br>1-2 = Normal Operation |
| JP0301 | **8051 Program Memory**<br>1-2 = Use External Program Memory.<br>2-3 = Use 8051's Internal Program Memory. |
| JP0302-<br>JP0304 | **USB Transaction Mode**<br>1-2 = Logic 0<br>Open = Logic 1<br><br>JP0302 is connected to 8051 I/O Port 1, Bit 6.<br>JP0303 is connected to 8051 I/O Port 3, Bit 4.<br>JP0304 is connected to 8051 I/O Port 3, Bit 5.<br><br>The local firmware tests these jumpers on start-up to determine the proper mode of operation. See the description below, and the `mode` command in Section 8.3.3 for more information. |
| JP0701 | **Device Remote Wakeup**<br>1-2 = Normal Operation<br>2-3 = Force wake up of suspended USB by forcing a 'ring in' on Serial Port 1. |
| JP0801 | **Reset**<br>Open = Normal Operation<br>1-2 = Board Reset |

## *3.2. Selecting USB Transaction Mode*

The NET2888 Evaluation Board has the ability to run in different 'modes' of operation.

Upon USB plug-in, the setting of JP0302 - JP0304 select a product feature mode for operation of the NET2888-EB. These jumpers only affect the state of the board on initial USB plug-in – they have no effect if changed while the board is running.

A different driver on the host PC will 'claim' the device depending on the jumper settings. The jumper settings are equivalent to the `mode` command in *LocalMon*, except that the jumpers will change which host driver claims the device.

NOTE: It is necessary to select mode '0' to test with the commands `rfi`, `tfi`, `rmb`, and `tmb` in *LocalMon*. Mode '0' is the only mode in which the user has control over the USB data flow. If mode '0' is not selected, packets coming from the host PC will be removed from the NET2888 automatically by the local firmware's interrupt routine.

Please refer to the table below for the proper jumper use:

| **Testing Mode** | **Equivalent** | **Jumper Settings** | | |
|---|---|---|---|---|
| (Host Driver Name) | **Mode #** | JP0302 | JP0303 | JP0304 |
| No Packet Processing (USBLOOP.SYS) | 0 | OFF | OFF | OFF |
| Loopback Testing (USBLOOP.SYS) | 1 | OFF | OFF | ON |
| COM Port Emulation (NC2888EB.SYS) | 3 | OFF | ON | ON |

To use Mode 3 (COM Port emulation), you must first follow the steps regarding "COM Port Emulation Software Installation," found in Section 4 of this manual.

Other settings are reserved for future product features.

### *3.3. 8051 I/O Port Settings*

| **Port 0** | **External Address (LSB) / Data Bus** |
|---|---|
| | |
| **Port 1** | **Miscellaneous Control /Status Signals** |
| P1.0 | ISO# output to NET2888 |
| P1.1 | USBOE# from NET2888 |
| P1.2 | DEVCFG# from NET2888 |
| P1.3 | SUSP# from NET2888 |
| P1.4 | DRQ from NET2888 |
| P1.5 | Shadow SRAM, maps SRAM into Program Memory |
| P1.6 | User Jumper JP0302 |
| P1.7 | LED, Logic 0 turns on LED |
| | |
| **Port 2** | **External Upper Address (MSB) Bus** |
| | |
| **Port 3** | **Miscellaneous Control /Status Signals** |
| P3.0 | 8051 serial port receive data (RXD) |
| P3.1 | 8051 serial port transmit data (TXD) |
| P3.2 | INT0, Interrupt from NET2888 (active low) |
| P3.3 | INT1, Interrupt from Super I/O (active low; IRQ3 or IRQ4 or IRQ7 or SUSPEND) |
| P3.4 | JP 0303 |
| P3.5 | JP 0304 |
| P3.6 | External Write Strobe |
| P3.7 | External Read Strobe |

### *3.4. Local Bus Memory Map*

| Address | Function |
|---|---|
| ***Program Space*** | |
| If Shadow# (P1.5) = 1 (disabled) | |
| 0x0000 - 0xFFFF | EPROM |
| If Shadow# (P1.5) = 0 (enabled) | |
| 0x0000 - 0x7FFF | EPROM |
| 0x8000 - 0xFFFF | SRAM |
| ***Data Space*** | |
| 0x0000 - 0x7FFF | SRAM data memory |
| 0x8000 - 0xBFFF | Super I/O (COM1, COM2, LPT1) |
| 0xC000 - 0xDFFF | NET2888 USB Device Controller |
| 0xE000 - 0xEFFF | NET2888 DMA acknowledge |
| 0xF000 - 0xFFFF | NET2888 DMA acknowledge with EOT |

# 4. COM Port Emulation / Demonstration Software

## *4.1. Installation*

In addition to the USBLOOP.SYS driver which performs loopback testing and allows generic transfers using *LocalMon* and *HostMon*, the evaluation board includes a pair of drivers to emulate a COM port over USB. These drivers, named NC2888EB.SYS and NCPORT.VXD, make Serial Port 1 (labeled JP0702) look like COM port 4 on the host PC. Any software package which addresses COM ports (e.g. HyperTerm) will transparently see Serial Port 1 on the NET2888-EB as a standard serial port, COM4.

This emulation mode requires interaction of both the host and the peripheral. On the peripheral-side, all data received by the NET 2888 from the Host PC will be sent out Serial Port 1 (labeled J0702 on the board). Similarly, data received on Serial Port 1 is sent to the Host PC via the Transmit FIFO. On the host, NCPORT and NC2888EB provide a mapping from Windows' Serial Port Communication Driver (VCOMM.VXD) to the top of Windows' USB stack layer (USBD.SYS).

Please note: You must use the jumpers JP0302-JP0304 to select 'Mode 3' on the NET2888-EB. See Section 3.2 for more information on Jumper settings.

**The steps required to install this driver are as follows:**

1. Power on the PC with Memphis OS running
2. Go to Control Panel, click on "**Add new hardware**" icon
3. When the "**Add New Hardware Wizard**" opens, click on ***Next***
4. At the second window, click on ***Next***
5. Check "No, the device isn't in the list", click on ***Next***
6. Check "No, I want to select a hardware from a list", click on ***Next***
7. Select the type of hardware you want to install
8. Highlight "**Ports (COM and LPT)**", then click on ***Next***
9. Select the manufacturer and model of your hardware:
   Under "Models" heading, highlight "**Communication Port"**
   Under "Manufacturer" heading, highlight "**Standard Port Types**"
10. Insert the Netchip distribution disk (Rev 0.5) into drive A
11. Click on ***Have Disk***. Select the COM Port emulation driver. You are done now.

**To check if installed correctly:**

1. Go back to Control Panel
2. Double click on "System"
3. Go to "Device Manager"
4. Click on "**Ports (COM & LPT)**". You should see listed "COM Port on NetChip NC2888EB (COM4)"

### *4.2. Limitations*

- The driver does not support handshaking lines (DSR, DTR, CTS, RTS).
- The driver does not support baud-rate changes. You must operate at 9600 baud.
- There are some unresolved plug and play issues with this driver. It is recommended that you boot up Memphis with the NET2888-EB already attached to the system if you have trouble using the emulation software.

# PART II
# The Host and Local Monitor

The host monitor (*HostMon*) and the peripheral local-side monitor (*LocalMon*) are built on the same code base, and operate almost identically. This section describes the operation of the host monitor and the local monitor. Most of this section applies to both programs. The term *NcMon* is used in this section to refer to both the *HostMon* and the *LocalMon* program. Features specific to each program are also described in this section.

# 5. NcMon Basics

## *5.1. Command Line*

The *NcMon* command line is similar to many command line programs—you type in commands, then press `ENTER`. Commands are case-sensitive.

One difference between the *NcMon* command line and command lines in other programs is that you can enter multiple commands (up to 80 characters) on one line. If there is any ambiguity between commands and parameters, enter a semicolon (`;`) between the commands.

An ambiguity may arise when a numeric parameter could be confused with a command. In hexadecimal, a digit includes characters "0" through "f". A numeric parameter, such "***db***," could be confused with the `db` "display bytes" command. For example, to display memory at 100, then do an input from 200 you can enter

```
d 100;i 200
```

The semicolon explicitly tells *NcMon* that "`I 200`" is a new command and not a numeric parameter for the display command, which can accept zero, one or two numeric parameters (refer to the section, "**d, db, dw, dl, dd**," for additional information). In this case, however, the semicolon is not necessary because "`i`" cannot be interpreted as a number. Therefore, entering

```
db 100 i 200
```

would be interpreted exactly the same as the first example.

The semicolon is required in this example:

```
db 100; db 200
```

because the second "`db`" could be interpreted as a numeric parameter for the first command.

### 5.1.1. User-Defined Variables

User-defined variables can be used as simple substitutes for obscure numbers, or they can be combined with expressions, numbers and commands to create more complex command sequences. Create (or modify) a variable by typing a *variable name*, followed by the equals sign (=), then followed by a *value*. For example, the value can be any legal combination of numbers, variables or expressions:

```
x = 1
x = x+1
```

You can also eliminate a variable by leaving the right-hand side empty:

```
x =
```

You can use virtually any name of any length for your variables.

**Note:** Commands and macros have priority over variables.

To view all variables, use the `vars` command (see section 6.2.2). The following example illustrates one way of using variables. The example outputs an incrementing value to a 16-bit port:

```
x = 0 [ow 100 x; x = x + 1; r 10]
```

Refer to the section 6.2.2 for additional information about expressions.

## *5.2. Expressions*

Expressions are arrangements of numbers, variables and operators that resolve to a simple number (see section, 6.2.1 for information about variables). *NcMon* enables you to work with expressions "on-the-fly"—anywhere you enter numbers, you can enter expressions.
Expressions can also even be evaluated—without a command—at the command line. Expressions are evaluated left to right, but can be overridden with parenthesis. The *NcMon* expression operators include monadic and dyadic operators, as described in the following sections.

### 5.2.1. Monadic Operators

Monadic operators include one's complement, two's complement, and so forth, as listed in Table 5-1:

**Table 5-1. Monadic Operators (where N represents a number, expression or variable)**

| Syntax | Operation | Example (Hex) | Result |
|---|---|---|---|
| ~N | Inverse of N | ~1 | Returns 0xfffffffe (One's complement) |
| -N | 0 minus N | -1 | Returns 0xffffffff (Two's complement) |
| +N | 0 plus N | +1 | Returns 0x00000001 (does not do anything!) |
| *N | Dereference Pointer N | *12345678 | Returns the value pointed to by 12345678 |

### 5.2.2. Dyadic Operators

Dyadic operators include addition, subtraction, and so forth, as listed in Table 5-2:

**Table 5-2. Dyadic operators (where N and M represent numbers, expressions or variables)**

| Syntax | Operation | Example (Hex) | Result |
|---|---|---|---|
| N+M | N plus M | 4+1 | Returns 0x00000005 |
| N-M | N minus M | 4-1 | Returns 0x00000003 |
| N*M | N multiplied by M | 7*3 | Returns 0x00000015 |
| N/M | N divided by M | 7/3 | Returns 0x00000002 |
| N%M | N modulo M | 7%3 | Returns 0x00000001 |
| N&M | N AND M | 7&3 | Returns 0x00000003 |
| N\|M | N OR M | 7\|3 | Returns 0x00000007 |
| N^M | N XOR M | 7^3 | Returns 0x00000004 |

## *5.3. Ranges*

Several *NcMon* commands require a full or partial range to be specified. The range includes a starting address and a byte count. For example, the byte count can be implied with an ending address or explicit:

```
db 4000 4020
```

which implies a count of 20 (4020-4000); whereas

```
db 4000 l 20
```

explicitly specifies a count of 20.

**Note:** "`l`" must be lowercase.

Both examples yield identical results. Commands that accept partial ranges use the starting address you type and substitute a default count. Some commands, such as the display commands, substitute both the starting address and count if you do not provide them.

## *5.4. Repeating Command Lines (Looping)*

A unique feature of *NcMon* is its repeat (`r`) command. The `r` command enables you to repeat all or part of a command line for a number of iterations you specify, or indefinitely. Further, you can nest repeat loops by using nesting the tokens, "`[`" and "`]`" (see section 6.2.3 for additional information).

## *5.5. Built-in Help*

*NcMon* provides built-in help messages for every command by typing `help` or `?` at the command prompt. You can get help for individual commands by typing `help` followed by the command name: `help d` or `? d`.

# 6. NcMon Commands

This section discusses the *NcMon* command set by category. Parameters listed with an *expr* substring can be numbers, variables, or expressions. Parameters listed within square braces ("[" and "]") are optional parameters, other parameters are necessary. Parameters seperated by a vertical bar (|) indicate that you must provide *one or the other* parameter, but *not both*, as in

```
command xexpr | yexpr
```

Parameters listed with ellipses (…) indicate that you can extend the command with more parameters of the same type.

```
command expr […]
```

## *6.1. Memory Manipulation Commands*

Memory manipulation commands allow you to display, modify, fill, search and compare memory in several intuitive and generally accepted ways. These commands are similar to the memory commands in other common debuggers. *HostMon* references memory as a flat, nonsegmented, 4 GB space. *HostMon* predefines two variables (ibuf and obuf) as the starting pointers to two 64 byte buffers. *LocalMon* references memory as several orthogonal flat 64K spaces defined by the 8051 CPU.

**Caution:** There are no restrictions on memory access (so you can easily crash your system).

Most of the memory manipulation commands accept range parameters. Ranges control the address limits of the command (refer to the section 6.2.7 for additional information). Display, enter and move have variations for byte, word and long word sizes. *NcMon* uses the appropriate byte, word or long word assembler instruction (hence, the appropriate bus cycle) for each variation.

The `d` and `e` commands adopt the size (byte, word or long word) of the most recently used display or enter command (that is, if you use an `el` command, subsequent `d` commands display long words).

**NOTE**: For commands that have different widths (e.g. `db`, `dw`, `dl`), only the byte version of the command is defined for *LocalMon*.

### 6.1.1. d, db, dw, dl, dd — Display memory

```
d [range]
db [range]
dw [range]
dl [range]
dd [range]
```

**Table 6-1. Display (d, db, dw, dl, dd) Command Input Parameter**

| Parameter | Description |
|---|---|
| *range* | Memory address and count (refer to section 6.2.7) |

This group of commands displays a range of memory. You can display byte, word, long word or double format (long word is the same as double). Display commands accept two, one, or zero parameters:

**Table 6-2. Display Command Input Parameter Information**

| No. | Parameter | Description |
|---|---|---|
| two | *range* (refer to the section, " Ranges") | Displays the specified memory range |
| one | *address only* | Displays memory starting at the address specified for the default count (0x80 bytes) |
| zero | — | Displays memory starting *after* the last address for the default count (0x80 bytes) |

*NcMon* displays the ASCII values of displayed memory in a separate column. This column displays in byte-order, regardless of the display size (byte, word, long word or double.)

Note: The ASCII column causes a second read of the displayed address. This may have consequences for certain hardware (the NET2888 auto-indexed registers, for example).

### 6.1.2. e, eb, ew, el, ed Enter Data into Memory

```
e addrexpr [INC incexpr] [valexpr …]
eb addrexpr [INC incexpr] [valexpr …]
ew addrexpr [INC incexpr] [valexpr …]
el addrexpr [INC incexpr] [valexpr …]
ed addrexpr [INC incexpr] [valexpr …]
```

**Table 6-3. Enter (e, eb, ew, el, ed) Command Input Parameters**

| Parameter | Description |
|---|---|
| *addrexpr* | Memory address to start entering |
| *incexpr* | Address increment |
| *valexpr* | Value to enter |

Use the Enter commands to modify—or to interactively examine and modify—memory. You can specify bytes, words, long words or doubles. (Long word is the same as double.)

Enter commands require an addrexpr parameter. If *valexpr* is not specified, *NcMon* enters Interactive-Enter mode. Interactive-Enter mode displays the value at the starting address and allows you type in a new value. You then have one of three choices:

1. Type a new value, then press ENTER or the SPACEBAR (the new value is actually written to memory when you press ENTER or the SPACEBAR).

2. Press the SPACEBAR.
3. Press ENTER.

Pressing the SPACEBAR continues Interactive-Enter mode at the next address, while pressing ENTER ends Interactive-Enter mode.

Pass at least one *valexpr* on the command line to have values entered in a noninteractive fashion.

Use *INC incexpr* when you want to enter values at noncontiguous addresses. For example, you may want to enter new values every 100 bytes, as in

```
e 123400 INC 100 a b c d
```

The *INC* specifier must be capitalized, and *incexpr* must be supplied. This feature can be applied in both Interactive-Enter and Noninteractive-Enter modes. The default for *INC* is 1.

### 6.1.3. f Fill memory range

```
f range fillexpr […]
```

**Table 6-4. Fill (f) Command Input Parameters**

| Parameter | Description |
|---|---|
| *range* | Fill address and count (refer to section 6.2.7) |
| *fillexpr* | Fill values (byte) |
| … | More fill values (bytes) |

You can use the f command to prepare a large portion of memory with data patterns. The pattern can be one or more bytes long. The fill values are repeated throughout the entire range.

### 6.1.4. m, mb, mw, ml, md Move memory range

```
m range destexpr
mb range destexpr
mw range destexpr
ml range destexpr
md range destexpr
```

**Table 6-5. Move (m, mb, mw, ml, md) Command Input Parameters**

| Parameter | Description |
|---|---|
| *range* | Source address and count (refer to the section, " Ranges") |
| *destexpr* | Destination address |

Use Move commands to move blocks of memory from one place to another. You can specify byte, word, long word or double variations. (Long word is the same as double.)

The byte variations (`m` and `mb`) are appropriate for most needs; the other variations are provided because they perform 16 and 32 bit data transfers at the assembler level. Unlike the `d` and `e` commands, the `m` command always moves bytes.

#### 6.1.5. c Compare memory range

```
c range addrexpr
```

**Table 6-6. Compare (c) Command Input Parameters**

| Parameter | Description |
|---|---|
| *range* | Left address and count (refer to the section, " Ranges") |
| *addrexpr* | Right address |

This command is useful for checking the result of a move or DMA. It performs a byte-by-byte comparison of data, starting at the left and right address locations. *NcMon* displays both addresses and both data bytes of any miscomparisons.

#### 6.1.6. s Search memory range

```
s range [expr […]] | ["text"]
```

**Table 6-7. Search (s) Command Input Parameters**

| Parameter | Description |
|---|---|
| *range* | Search address and count (refer to the section, " Ranges") |
| *expr* | First byte of search pattern |
| … | Subsequent bytes of search pattern |
| *"text"* | Text pattern being searched |

This command looks for a data pattern. You can search for an ASCII string or a data pattern that is one or more bytes in length. The address of any matching pattern within the range is printed. Text must be typed between quote (" ") characters. The search is case-sensitive.

### *6.2. NcMon Internal Commands*

The following sections discuss the *NcMon* internal commands.

#### 6.2.1. vars Show variables

```
vars
```

This command displays all variables. Refer to the section 5.1.1 for information about defining variables.

#### 6.2.2. expr Evaluate expression

```
expr valexpr
```

**Table 6-8. Expressions (`expr`) Command Input Parameter**

| Parameter | Description |
|---|---|
| *valexpr* | Expression to be evaluated |

This command evaluates *valexpr* and displays the result in hexadecimal and signed-decimal format.

**Note:** Expressions can be evaluated without this (or any) command. Simply entering an expression at the command prompt will show the (hexadecimal) value of the expression.

### 6.2.3. r Repeat command line

```
r [countexpr]
```

**Table 6-9. Repeat (`r`) Command Input Parameter**

| Parameter | Description |
|---|---|
| *countexpr* | Iteration count |

This command repeats all or part of a command line indefinitely or for a specific number of iterations. In its simplest application, you append the `r` command to the end of the command line; *NcMon* repeats the entire command line over and over until you press a key:

```
ob 100 ab; r
```

Adding a *countexpr* variable limits the number of iterations:

```
ob 100 ab; r 10
```

You can use nesting tokens, "`[`" and "`]`", to repeat part of a command line:

```
ib 200 [ob 100 ab; r 10]
```

This example inputs one byte from port 200, then outputs `ab` to port 100 16 times. An absence of nesting tokens tells *NcMon* to repeat, starting at the beginning of the line. There is no limit to the number of nesting tokens you can apply. For example:

```
d 1000 [ib 200 [ob 100 ab; r 10]]r 5
```

### 6.2.4. echo Echo a string to screen

```
echo text
```

**Table 6-10. `echo` Command Input Parameter**

| Parameter | Description |
|---|---|
| *text* | Any text up to the point where you press ENTER |

This command echoes the remainder of the command line. Use this command to document a *NcMon* command file (refer to the section 7.2.4 for information on saving command files).

### 6.2.5. wait Wait for some time

```
wait countexpr
```

**Table 6-11. `wait` Command Input Parameter**

| Parameter | Description |
|---|---|
| *countexpr* | Number of cycles to wait |

This command waits while the processor down-counts *countexpr*. The wait loop is uncalibrated; *NcMon* simply "sits" in a software loop until the until *countexpr* reaches zero. The longest wait is obtained by passing "0" as the parameter. This is equivalent to wait 100000000. There is no way to break out of a wait, so be careful using a large *countexpr* value.

### 6.2.6. base Change the default radix

```
base [baseexpr]
```

**Table 6-12. `base` Command Input Parameter**

| Parameter | Description |
|---|---|
| *baseexpr* | New radix (base 16 max) |

This command sets the *NcMon* radix to a new base. If *baseexpr* is a number (not a variable or an expression), it must be in base 10. The highest base (and the default base) of *NcMon* is base 16. Entering numeric parameters to a command using digits greater than the radix (but less than F) generates an error message. However, the command will continue, and the offending digits are truncated to zero.

Use `base` with no parameters to determine the current radix of *NcMon*.

### 6.2.7. range

```
range
```

This command does nothing! It is simply a placeholder for the *NcMon* built-in help for using ranges. Refer to the Section 5.3 for information about specifying ranges.

### 6.2.8. environment

```
environment
```

This command does nothing! It is simply a placeholder for the *NcMon* online help for the environment variable.

### 6.2.9. cmd_edit

```
cmd_edit
```

This command does nothing! It is simply a placeholder for the *NcMon* online help for editing command lines.

### 6.2.10. help, h, ? Display help

```
help [command]
h [command]
? [command]
```

**Table 6-13. `help`, `h`, `?` Command Input Parameter**

| Parameter | Description |
|---|---|
| command | Command name |

These commands display online help for *NcMon* commands. If no parameter is specified, these commands print a complete list of *NcMon* commands. Enter an *NcMon* command as a parameter to display the online help message for that command. Enter an asterisk (`*`) as a parameter to display the online help messages for all commands.

The help text of a command may display a parameter within square braces ("`[`" and "`]`"). Such a parameter is optional for the command. The help text of a command may display ellipses (...) following a parameter. The ellipses indicate that more parameters of the same type can follow the first parameter.

# 7. Features specific to HostMon

## *7.1. USB Commands*

### 7.1.1. closeall — Close handles to devices on USB.

Force *HostMon* to close its handles to all devices attached to the USB.

### 7.1.2. ebcheck — Check the NET2888-EB.

This is not really a command at all, but is instead a pre-defined macro that performs loopback testing on an installed NET2888-EB.

### 7.1.3. findall — Find all devices attached to USB.

Find and list all devices currently attached to the USB. This is a necessary step before performing other USB-related commands. Only fully enumerated devices (i.e. those that have successfully been associated with a host driver) will appear on this list. If you un-plug and re-plug the NET2888-EB, you need to re-run this command so that *HostMon* can get the new USB address of the board.

### 7.1.4. gcd — Get Configuration Descriptor

```
gcd
```

Issues a USB standard request for the "configuration descriptor." A USB request for a configuration descriptor causes a peripheral to return the configuration descriptor, interface descriptor(s), and endpoint descriptor(s). After the device returns these descriptors, the returned values are compared against expected values. The returned values are reported to the user on-screen.

### 7.1.5. gdd — Get Device Descriptor

```
gdd
```

Issues a USB standard request for the "device descriptor." After the device returns the device descriptor, the values returned are compared against expected values. The returned values are reported to the user on-screen.

### 7.1.6. ifi — Input from FIFO

This command reads up to 64 bytes from the NET2888 Transmit FIFO (endpoint 4) and places it in *ibuf*.

### 7.1.7. imb

This command reads 8 bytes from the NET2888 Transmit Mailbox (endpoint 2) and places it in *ibuf*.

### 7.1.8. loopfi — FIFO loopback

Performs a single loopback iteration through the receive and transmit FIFOs on the NET2888, and compares the data sent to the data received. NOTE: *LocalMon* must be running in loopback mode. Data is sent from *obuf* and received into *ibuf*.

#### 7.1.9. loopmb Mailbox loopback

Performs a single loopback iteration through the receive and transmit mailboxes on the NET2888, and compares the data sent to the data received. NOTE: *LocalMon* must be running in loopback mode. Data is sent from *obuf* and received into *ibuf*.

#### 7.1.10. ofi Output to FIFO

```
ofi [range]
```

**Table 7-1. `ofi` Command Input Parameter**

| Parameter | Description |
|---|---|
| *range* | Memory address and count (refer to section 6.2.7) |

This command sends data to the NET2888 Receive FIFO (endpoint 3). Without any parameters, `ofi` will send a 64-byte packet from *obuf*. The optional parameter specifies a memory range to use for the data to send. This also allows the user to specify a range of less than 64 bytes, so that a shorter packet may be sent to the NET2888's receive FIFO. `ofi` without any parameters is equivalent to `ofi obuf l 40`

#### 7.1.11. omb Output to Mailbox

```
omb [range]
```

**Table 7-2. `omb` Command Input Parameter**

| Parameter | Description |
|---|---|
| *range* | Memory address and count (refer to section 6.2.7) |

This command sends data to the NET2888 Receive Mailbox (endpoint 1). Without any parameters, `omb` will send a 8-byte packet from *obuf*. The optional parameter specifies a memory range to use for the data to send. This also allows the user to specify a range of less than 8 bytes, so that a shorter packet may be sent to the NET2888's receive mailbox. `omb` without any parameters is equivalent to `omb obuf l 8`

### *7.2. Internal Commands*

#### 7.2.1. define Define a macro

```
define macname [macbody]
```

**Table 7-3. Define a Macro (`define`) Command Input Parameters**

| Parameter | Description |
|---|---|
| *macname* | Macro name |
| *macbody* | Macro body |

This command defines (or deletes) a macro. The first parameter names the macro, while the remainder of the line is stored as the macro body. The macro body can include commands, variables and other macros.

**Note:** The macro body is *not* checked for validity.

**Caution:** Referencing the same macro within a macro works; however, this will probably overflow the *HostMon* stack!

To execute a macro, type the macro name anywhere you would normally type a command.
Once a macro is defined, you can delete it by using the `define` command with the macro name as the only parameter. A macro can be redefined by entering the macro name, followed by a new macro body.

*HostMon* scans the built-in command list before the macro list. So, if your macro name matches a built-in command, the command will be executed, and your macro will be ignored. There is room for about 25 macros at 80 characters per macro. Macros can be saved to a file for future use (refer to section 7.2.4 for additional information).

### 7.2.2. macs Show Macros

```
macs
```

This command displays all macros. Refer to the previous section for defining macros.

### 7.2.3. quit, q Quit HostMon

This command quits *HostMon*.

### 7.2.4. read Read from disk

```
read [filename]
```

**Table 7-4. `read` Command Input Parameter**

| Parameter | Description |
|---|---|
| *filename* | Name of command file |

This command reads and executes a *HostMon* command file. *HostMon* reads characters from the file as if they are typed-in command lines. If no parameter is specified, `read` uses `SAVE.MON` as the default filename.

The companion to `read` is the `save` command. The `save` command saves the *HostMon* macros and variables in a format ready for the `read` command.

### 7.2.5. save Save to disk

```
save [filename]
```

**Table 7-5. `save` Command Input Parameter**

| Parameter | Description |
|---|---|
| *`filename`* | Name of command file |

This command saves current variables and macros to the specified filename. The file is saved as simple, editable text. The variables and macros are saved as *HostMon* commands—you can edit the file, even add other *HostMon* commands, then execute the file at any time using the `read` command. If no parameter is specified, `save` uses `SAVE.MON` as the default filename.

## *7.3. Input and Output Commands*

The *HostMon* input and output commands provide unrestricted port I/O. Variations for byte, word or long word port I/O are provided. The assembler instruction used for the port transfer is appropriate for the I/O width (byte, word or long word) you specify.

### 7.3.1. i, ib, iw, il, id Input from port

```
i [range]
ib [range]
iw [range]
il [range]
id [range]
```

**Table 7-6. Input (`i`, `ib`, `iw`, `il`, `id`) Command Input Parameter**

| Parameter | Description |
|---|---|
| *`range`* | Port address and count (refer to the section, " Ranges") |

The Input group of commands reads and displays from a range of port addresses. You can display in byte, word, long word or double format. (Long word is the same as double.)
Input commands accept two, one or zero parameters:

**Table 7-7. Input Command Input Parameter Information**

| No. | Parameter | Description |
|---|---|---|
| two | *`range`* (refer to Section 5.3) | Specified range of ports is read and displayed |
| one | *`address only`* | Port address specified is read and displayed |
| zero | — | Port address starting *after* the last port address read is read and displayed |

The simplest usage is to specify the port address (one parameter), as in

```
iw 100
```

The `i` command adopts the size (byte, word or long word) of the most recently used input command (that is, if you just used the `il` command, the `i` command reads and displays long words).

*HostMon* displays the ASCII values of the port data in a separate column. This column displays in byte-order, regardless of the size specified (byte, word, long word or double.)

### 7.3.2. ob, ow, ol, od — Output

```
ob portexpr valexpr
ow portexpr valexpr
ol portexpr valexpr
od portexpr valexpr
```

**Table 7-8. Output (`ob`, `ow`, `ol`, `od`) Command Input Parameters**

| Parameter | Description |
|---|---|
| *portexpr* | Port address |
| *valexpr* | Value to be written |

Use an Output command to write data to ports. You can write bytes, words, long words or doubles. (Long word is the same as double.)

## *7.4. Serial Port Input and Output Commands*

*HostMon* includes several commands to allow a user to transmit and receive data through one of the host's COM ports.

### 7.4.1. open — Open serial communications port

```
open comport
```

**Table 7-9. `open` Command Input Parameters**

| Parameter | Description |
|---|---|
| *comport* | COM port number (1, 2, 3, etc.) |

This command will 'open' the COM port specified for data transmission and reception. After the port is opened, the `enter` and `write` commands may be used to send data through this COM port, and *HostMon* will monitor the port for received data. Received data will be stored and displayed in the format of Display memory command.

Only one COM port can be open at once; you must use the `close` command to close a port before another COM port may be opened.

### 7.4.2. close — Close serial communications port

```
close
```

This command closes the serial communications port opened with the `open` command.

### 7.4.3. enter — Enter data to communications port

```
enter  valexpr […]
```

**Table 7-10. `enter` Command Input Parameters**

| Parameter | Description |
|---|---|
| *valexpr* | Value to enter |

This command enters hex bytes to the communications port opened with the `open` command.

#### 7.4.4. Write Write buffer to communications port

`write`

This command writes the 64-byte *obuf* to the communications port opened with the `open` command.

## *7.5. Macros*

Macros enable you to create a complex sequence of commands, and call it by a name specified by you. To execute the complex sequence of commands, use the macro name instead. Macros are convenient if you find yourself using a complex command sequence, over and over again. Macros, as well as variables, can be saved to a file, then restored in a future session, using the `save` and `read` commands.

Note: Due to memory restrictions, macros have been disabled in *LocalMon*.

## *7.6. When HostMon Starts…*

*HostMon* takes the following actions when it starts.

1. *HostMon* displays release notes, if any exist. *Read these notes!* They may contain late-breaking changes to the program.
2. Performs a **findall** command to list the devices currently recognized on the USB.
3. Displays the *HostMon* prompt (`&`).
4. *HostMon* is ready for user input. Type commands, variables or expressions, then press `ENTER`.

## *7.7. Scrolling and Editing Command Lines*

*HostMon* allows you to scroll to previous command lines you have entered. To scroll, use the up (↑) and down (↓) arrow keys. You can edit any line by using the left (←) and right (→) arrow keys, `BACKSPACE`, and `ESC`. *HostMon* defaults to Overwrite mode. Press the `INSERT` key to toggle between Overwrite and Insert modes.

# 8. Features specific to LocalMon

## *8.1. Starting LocalMon*

The local monitor (*LocalMon*) is a program which runs on the Evaluation Board's 8051 CPU. It is a full-featured monitor program which includes the ability to perform USB packet transfers, USB loopback mode, and USB to Serial Port pass-through.

To use the evaluation board's local monitor, follow the steps below.

1. Connect the NET2888-EB Serial Port 0 (the serial port closest to the USB connector) to a host computer's serial port, say, COM1. You must use a null modem cable to make this connection, since both the NET2888-EB and the computer's serial ports are DTE ports.
2. Use a serial terminal emulation program on the host computer (e.g. Microsoft's HyperTerm) to connect at 9600 baud with 8 data bits, no parity, and 1 stop bit (8-N-1). Also, set flow control to 'none'.
3. Connect the NET2888-EB's USB port to the host computer. You should see *LocalMon*'s startup message and prompt ("&") on your terminal emulator. If not, try pressing ENTER to see if the NET2888-EB responds. If it doesn't, review your connection settings.
4. Read the rest of this chapter about *LocalMon*. Remember you can also type `help` at the *LocalMon* prompt to see what commands are available.

## *8.2. General Operation*

*LocalMon* has the ability to operate in several different 'modes', which affect how data received and/or transmitted over USB will be processed by the NET2888-EB. These modes are set with the 'mode' command, described below. The default startup mode for the NET2888-EB depends on the setting of Jumpers JP0302 - JP0304. See also the jumper description in Section 3.2.

## *8.3. USB Commands*

### 8.3.1. encr — Display/Modify NET2888 extended regs

```
encr
encr [eregnum]
encr eregnum [newval]
```

| Parameter | Description |
|---|---|
| *eregnum* | NetChip extended register number (00 - FF) |
| *newval* | New value to assign to register |

The NET2888 contains a number of extended registers which are accessed with the EXTIDX and EXTDATA base registers. `encr` provides a convenient method to access these registers with a single command from the *LocalMon* prompt.

If `encr` is entered without any parameters, all NET2888 extended registers and their contents are displayed on screen. With a single parameter, a single register's contents are displayed. With two parameters, a new value is written to a register.

### 8.3.2. loop — Enter loopback mode

```
loop
```

This command is provided for backward compatibility, and is equivalent to the command `mode 1`. See the description of the `mode` command, below.

### 8.3.3. mode — Change USB Transaction Mode

```
mode modenum
```

| Parameter | Description |
|---|---|
| *modenum* | Set mode number for USB transaction processing |

This command sets the current mode of operation of the NET2888-EB, which determines how the NET2888-EB performs USB transactions. See the table below.

***NET2888-EB modes of operation***

| *Modenum* Value | Mode Description |
|---|---|
| *0* | **No processing**. Received data remains in the NET2888's receive FIFO and receive mailbox registers. Data may be received and transmitted with commands including **rfi**, **tfi**, **tmb**, and **rmb** commands in *LocalMon*, and **ofi**, **ifi**, **omb**, and **imb** in *HostMon*. |
| *1* | **Loopback mode**. Data received in the NET2888's receive FIFO (EP3) is immediately placed in the transmit FIFO (EP4). Data received in the NET2888's receive mailbox (EP1) is immediately transferred to the transmit mailbox (EP4). Loopback mode is supported in *HostMon* with commands such as **loopmb** and **loopfi**. |
| *2* | **Not supported**. |
| *3* | **Serial port passthrough mode**. Data received in the NET2888's receive FIFO (EP3) is transmitted through Serial Port 1 on the NET2888-EB. Data received by Serial Port 1 is placed in the NET2888's transmit FIFO (EP4) for transmission. With an appropriate host driver, this mode may be used to emulate a Serial port over USB with the NET2888-EB. |

These operations are performed in the background as an interrupt service routine, while *LocalMon* is running. The default 'start-up' mode of operation depends on the setting of Jumper JP0302 - JP0304 on the NET2888-EB. See Section 3.2 for proper jumper settings.

### 8.3.4. nci — Initialize the NET2888

```
nci
```

Re-initialize the IRQSTAT1, IRQENB1, EP1IDX, EP2IDX, EP3STAT, EP4STAT, USBSTAT, and DCTL registers with the values loaded on board reset.

### 8.3.5. ncr Display / Modify NET2888 registers

```
ncr [regnum]
ncr regnum [newval]
```

| Parameter | Description |
|---|---|
| *regnum* | NetChip register number (00 - 1F) |
| *newval* | New value to assign to register |

If `ncr` is entered without any parameters, all NET2888 registers and their contents are displayed on screen. With a single parameter, a single register's contents are displayed. With two parameters, a new value is written to a register.

### 8.3.6. p1 Display / Modify 8051 CPU Port 1

```
p1 [newval]
```

| Parameter | Description |
|---|---|
| *newval* | New value to assign to register |

Edit or display the contents of the 8051 CPU I/O Port 1. This port is connected to various signals on the NET2888-EB. If `p1` is entered without any parameters, the current value of p1 is displayed. With a single parameter (*newval*), the value of the parameter is loaded into port 1. Port bits configured as inputs should be written with a '1'.

The description of each port bit is shown below.

**8051 CPU Port 1 Bits**

| Port Bit | Direction | Description |
|---|---|---|
| *7* | Output | **Board LED**. 0 = On, 1 = Off |
| *6* | Input | **Jumper JP0302**. 1 = Not connected, 0 = Connected. |
| *5* | Output | **Shadow# mode**. 0 = NET2888-EB memory map in Shadow RAM mode (see Section 3.4) |
| *4* | Input | **NET2888 DRQ pin**. 1 = Outstanding DMA request. |
| *3* | Input | **NET2888 SUSP# pin**. 0 = NET2888 is suspended. |
| *2* | Input | **NET2888 DEVCFG# pin**. 0 = NET2888 has been enumerated by USB Host. |
| *1* | Input | **NET2888 USBOE# pin**. 0 = NET2888 is currently driving USB data lines. |
| *0* | Output | **NET2888 ISO# pin**. 1 = FIFOs in Bulk mode, 0 = FIFOs in Isochronous mode. |

### 8.3.7. rfi Receive FIFO

```
rfi [range]
```

| Parameter | Description |
|---|---|
| *range* | New value to assign to register |

Read data from the NET2888 Receive FIFO (endpoint 3) into local memory. If no memory range is specified, the data will be read into location *rbuf*.

### 8.3.8. rmb Receive Mailbox

```
rmb [range]
```

Read data from the NET2888 Receive Mailbox (endpoint 1) into local memory. If no memory range is specified, the data will be read into location *rbuf*.

### 8.3.9. term Not Supported

Not supported. This command is equivalent to `mode 2`.

### 8.3.10. tfi Transmit FIFO

```
tfi [range]
```

Write data from local memory into the NET2888 Transmit FIFO (endpoint 4) to prepare it for transmission. If no memory range is specified, data will be read from location *tbuf*.

### 8.3.11. tmb Transmit Mailbox

```
tmb [range]
```

Write data from local memory into the NET2888 Transmit Mailbox (endpoint 2) to prepare it for transmission. If no memory range is specified, data will be read from location *tbuf*.

# 9. Source Code

## *9.1. Host Monitor (HOSTMON.EXE)*

The source code for *HostMon* is provided on the floppy disk in the \SRC\HOSTMON directory. The application interfaces with Microsoft's USBLOOP.SYS driver written for the WDM interface. The code is commented and may be used as a model for development of a host application. The host monitor is by no means a complete application, but does show how to perform basic transfers across the USB.

## *9.2. NET 2888 Local-Bus Monitor (LOCALMON)*

The source code for the NET2888 local monitor is provided on the floppy disk in the \SRC\LOCALMON directory. The source code is written in C, for the 80x51 CPU platform. The code is commented and may be used as a model for development of software for a peripheral to interface with the NET2888.

Also, if you have access to a PROM programmer, you may wish to modify this code to provide additional functionality, such as emulating a device in software, or enabling other transfers through the parallel / serial ports.

## *9.3. Host Driver (USBLOOP.SYS)*

The binary version of the host driver is provided on the floppy disk. The source code to this driver is available on Microsoft's WDM DDK CD. This driver provides an interface between the USB layers of the operating system, and an application such as *HostMon*. Studying this driver may provide insight into the development of a host driver.