

INTEGRATING NETROM™ WITH CROSSVIEW®

INTRODUCTION

The NetROM EPROM emulator provides you with a networked debug path to your target board. In combination with BSO/Tasking's debugger CrossView you will have a development solution that allows you to debug your application on your workstation or PC while your target is connected to your network at any remote place. Your target does not need to have network hardware for this. The debug monitor that runs on the target to control the execution for CrossView may need special configuration for NetROM. The debugger and the NetROM emulator use the TCP/IP network protocol to communicate. If you have a serial port on your target which can be connected to the NetROM, you can use the standard debug monitor, which uses the serial interface for I/O. The NetROM translates the information on the serial line to TCP/IP packages which are sent to the debugger. The advantage of using the serial hookup is that the monitor does not need any modification. The disadvantage is that the download speed remains limited to the line speed of the serial interface regardless of the speed of the network interface between the debugger and NetROM.

Figure 1 shows the typical setup of a target board and the NetROM using the serial interface. Note that you can have multiple windows connected to the NetROM. The MON program is a very simple terminal emulation program that is part of the CrossView product. This program has a TCP/IP interface that allows you to specify the remote node and the service port number. (Port 1235 is the default debug port on the NetROM. For more information, see the NetROM User's Guide.)

Before you can start the network version of the debugger you have to make sure that the NetROM properly booted and the target monitor has been loaded successfully. The NetROM reads a bootfile from a RARP/BOOTP server, depending on your network setup. Please follow the NetROM User's Guide to prepare your network software. The NetROM expects a file with boot parameters. For a RARP boot solution the filename is derived from the Internet address of the NetROM by translating the address: e.g. for Internet address 192.0.0.210 the NetROM attempts to read the file C00000D2. (0xC0 = 192 0xD2=210) To configure the NetROM for a board that uses a single 256K EPROM (27c020) the following boot parameter file is a typical example:

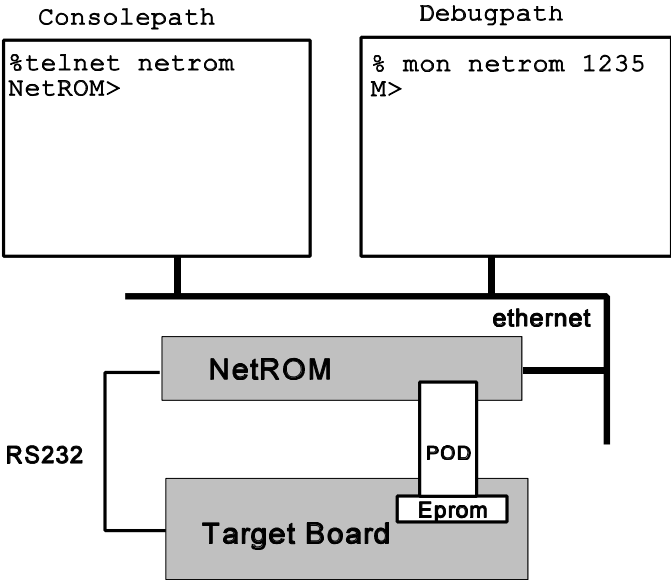


Figure 1. NetROM connected with serial interface to target

```

begin
  setenv    romtype      27c020
  setenv    wordsize     8
  setenv    romcount     1
  setenv    loadpath     /tftpboot
  setenv    loadfile     mon.rom
  setenv    host         192.0.0.209
  setenv    groupaddr    bfc00000
  setenv    groupwrite   readonly
  setenv    podorder     1
  setenv    filetype     srecord
  setenv    debugpath    serial
  setenv    consolepath  serial
  tgtreset
  stty      target       noxon
  set       emulate      on
  set       debugecho    off
  set       pgconfig 0 27c020 bfc00000 0 readonly
  newimage /tftpboot/mon.rom type=srecord base=bfc00000
end

```

Note that there are two parameters in the NetROM that determine where output will be sent to *debugpath* and *consolepath*. For the setup of figure 1 both variables have to be set to *serial*. The filename of the monitor image that will be downloaded to your target, *mon.rom* is just an example filename. It is not one of the images of the monitor in the CrossView product. Which one you select depends on the board you use. Make sure that you select an image that has been prepared to run from an EPROM (not the *ma* version), since the NetROM replaces the EPROM on your target.

You can check that the NetROM is up and running by starting a telnet session, shown in the top left box in figure 1. If you do not get the NetROM> prompt then probably your network has not been configured properly. If you have a connection with the NetROM you can check that it has read the boot parameters correctly by typing the command *printenv*. If these NetROM parameters do not resemble the settings you configured in the bootfile, check that this file is accessible via the network for foreign users. (Enable 'world reading on both the file and the directory') If you are sure that the right image has been loaded and *debugpath* has been set to *serial*, you can start the mon program on port 1235. After resetting the target you should see the monitor prompt. After this you can disconnect the mon program (^D) and start the network version of CrossView. On the first invocation of the debugger you'll have to select the item 'Emulator Communication Setup' in the file menu. This opens a dialog that allows you to enable the network for communication with the NetROM. Select *TCP/IP Communication*. Enter the name your network administrator entered in the hostname database for the NetROM. The port number you can select is 1235 (the default). Note that the number you enter has to match the value of the NetROM variable *debugport*. E.g. if you add the line '*setenv debugport 3003*' to the bootfile of the NetROM, you can invoke the MON program without specifying a port number (*mon<netrom>*) uses the default port 3003 and in the debugger you have to select port 3003.

DEBUGGING WITHOUT A SERIAL PORT ON YOUR TARGET

Instead of using the serial connection between the NetROM and the target, you can use the emulation POD to establish a communication path between CrossView and the debug monitor. For this the debug monitor has to be recompiled: the serial port driver has to be replaced with a the virtual port driver which accesses the emulation memory in POD0 on the NetROM. When you configure the NetROM you have to specify that the debug information will be sent through POD0. You have to set the variable *debugpath* in the boot configuration file: *setenv debugpath readaddr*

The debug monitor source directory contains a file *m_netrom.c*. This file contains the functions that replace the serial port driver functions. The NetROM supports two techniques to write the emulation memory in POD0 for communication: by detecting write requests to the EPROM and a write-by-read

protocol. By reading specific buffer addresses in the EPROM, the NetROM interprets this as a write request. This may be necessary because your EPROM may not have the write strobe connected. Thus the NetROM would not be able to detect the write requests. Even if you do have a write strobe connected the write-by-read protocol will work. For this reason `m_netrom.c` implements this protocol. (Which matches the `readaddr` configuration parameter.)

CONFIGURING THE DEBUG MONITOR

The debug monitor directory contains a makefile. This makefile has been written for the make utility `mk3`, which is part of the R3000/R4000 compiler package. Note that this makefile will not execute properly when used with a regular make, since the makefile contains controls like `ifdef`, `else`, `endif`, which are not supported by a regular make utility. These controls have been added in `mk3` to make conditional compilation easier. For NetROM support you have to have version 3.1 of the debug monitor or higher. Older versions do not yet have support for NetROM's virtual uart driver.

Configuring the monitor for NetROM requires you to regenerate the ROM version of the monitor with the module `m_netrom.c` instead of the serial driver. `m_netrom.c` calls functions which access `POD0` for I/O. These functions are in the module `dptarget.c`. Since only some of the functions in `dptarget.c` are used, you can shrink the object size of the monitor by removing the unused functions from `dptarget.c`. The reason that this has not been done already is that it makes integration of newer revisions of `dptarget.c` more difficult. This module is supplied by XLNT Designs, Inc., the manufacturer of NetROM. The following command rebuilds the monitor image `rocket.rom` with NetROM support:

```
mk3 "NETROM=1" rocket.rom
```

Notice that some modules are recompiled using the flag `'-DPOLL'`. Normally the monitor operates on an interrupt basis. Characters sent to the target cause an interrupt, for which the monitor has a handler that fills the receive buffer. The advantage of using interrupted I/O is that if your application runs and does not hit any breakpoint, the monitor can still regain control if you send a break character (`^C`, 003). The interrupt caused by the UART broke the execution of your program. When the interrupt handler detects the break character it returns to the monitor and not to your application. When the EPROM socket and the emulation memory in `POD0` are used for communication, hardware interrupts can no longer be generated. Thus the monitor has to poll the NetROM regularly if a character has been received. Normally you will not notice any difference when using polled I/O instead of interrupted I/O. However, you can no longer break execution when your program runs without returning to the monitor, since this locks out the polling code of incoming characters.

It is possible to have interrupted I/O when using the virtual uart interface: NetROM supports up to 8 control signals through a pin connector on the front of the box. You can configure one of these pins to serve as an interrupt line for incoming packets on the NetROM. The command:

```
set tgtctl 1 on rx
```

will cause pin 1 to be asserted when a packet is received.

```
set tgtctl 2 off rx ack
```

will de-assert pin 2 when a packet is received when the previous packet has been acknowledged. (This prevents nested interrupts)

You are advised to connect the pin to a latched interrupt on the target.

Note that if you want to use the interrupt signal from NetROM you have to modify the exception handler in the monitor. Currently the exception handler only services interrupts generated by the serial interface. If you connect the NetROM to one of the other interrupt inputs of the processor, the exception handler has

to be modified.

The module `m_netrom.c` does not support acknowledgment of received packets, which is necessary if you configure the control signal with the `ack` option. If you compile without `-DPOLL` the receive function in `m_netrom.c` (`io_rx`) will no longer be called from the main monitor loop but from the exception handler. If you have configured the NetROM signal with the `ack` option, you have to add a call to the function `ra_rx_intr_ack()` in `dptarget.c` to acknowledge the receipt of data and allow the next interrupt.

REMOTE TARGET RESET

You can reset your target from a remote location by connecting the R-control signal on the front connector of the NetROM to the reset line on your target. If you enter the command `'tgtreset'` (in the telnet session that gives you access to the NetROM> prompt) NetROM generates a reset pulse on the R control pin.

Information furnished is believed to be accurate and reliable. However, BSO/Tasking assumes no responsibility for the consequences of use of such information nor for any infringement of patents or rights of third parties which may result from its use. No license is granted by implication or otherwise under any patent or patent rights of BSO/Tasking. Specifications mentioned in this publication are subject to change without notice. This publication supersedes and replaces all information previously supplied. BSO/Tasking products are not authorized for use as critical components in life support devices or systems without the express written approval of BSO/Tasking.

© 1995 Tasking Software BV - All Rights Reserved

Boston Systems Office, BSO, Boston Systems Office/TASKING, BSO/TASKING and CrossView are registered trademarks of Boston Systems Office Inc and Tasking Software BV.