Home

OS-9® for Motorola® Compact PCI Board Guide

Version 4.7

RadiSys
THE POWER OF WE

www.radisys.com

Revision A • July 2006

Copyright and publication information

This manual reflects version 4.7 of Microware OS-9.

Disclaimer

Reproduction notice

July 2006

Table of Contents

Chapter 2: Board Specific Reference 41

Appendix A: Board Specific Modules 61

Chapter 1: Installing and Configuring OS-9®

The chapter describes installing and configuring OS-9® on the Motorola® MCP750 Compact PCI reference board. It includes the following sections:

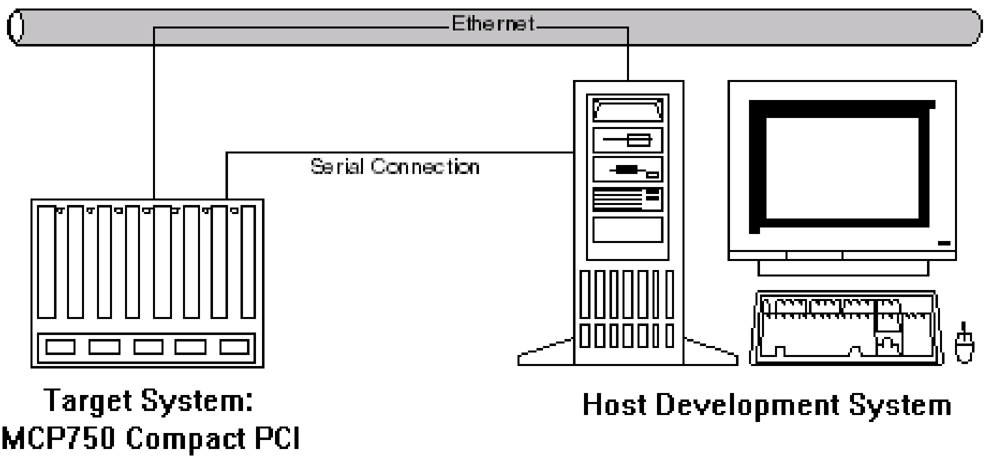
RadiSys.

MICROWARE SOFTWARE

Development Environment Overview

Figure 1-1 shows a typical development environment for the Motorola MCP750 Compact PCI reference board. The components shown include the minimum required to enable OS-9 to run on PowerPC™.

Figure 1-1 MCP750 Development Environment



Requirements and Compatibility

Note

Before you begin these sections, complete the following tasks:

- Install ***Microware OS-9 for PowerPC*** on your host system.
- Install the target board into its enclosure and connect it to the desired peripherals and equipment.
- Boot the target board to the `PPC1-Bug>` command prompt for the Motorola PowerPC debugger.
- Read the Motorola MCP target board hardware documentation.

Host Hardware Requirements (PC Compatible)

Your host PC must meet the following minimum requirements:

- Windows 95, 98, ME, 2000, or NT
- 300-400 MB of free disk space
- an Ethernet network card
- 16MB of RAM (32MB is recommended)
- one free serial port

Host Software Requirements (PC Compatible)

Your host PC must have the following applications:

- a terminal emulation program (such as Hyperterminal, which comes with Microsoft Windows 95, Windows 98, and Windows NT)
- the `TFTPSERVERPro` server application for downloading the OS-9 ROM image to the MCP750 target

  This application is included with Microware OS-9 for PowerPC and is loaded onto your host PC during the CD-ROM installation process.

Target Hardware Requirements

Your MCP750 reference board requires the following hardware:

- enclosure or chassis with power supply
- an RS-232 null modem serial cable
- compact FLASH memory card
- RAM300 memory mezzanine

Target Hardware Setup

The following section details setting up the target board.

Setting the Switches on the Target Board

You must modify the jumper settings for Flash. When programming the Flash system, you must have the Flash bank B (1MB) area enabled. This enables programming of the Flash bank A (4MB or 8MB) section.

For More Information

Refer to the appropriate ***Installation and Use*** and ***Programmer's Guide*** documents from Motorola for more information about programming the Flash system on your reference board. You can access these documents directly from your web browser by opening the following url:

`http://mcg.motorola.com`

Connecting the Target to the Host

Use an RS-232 null modem cable to connect the target to the serial port of your host system. Depending on your host PC, you may need either a straight or reversed serial cable.

With the target system powered off, connect the serial cable to the COM1 port on the reference board.

You must also connect the host and target systems to a network to use TFTP.

Complete the following steps to connect the target to the host:

Step 1. Connect the other end of the serial cable to the desired communication (COM) port on the host system.

Step 2. On the Windows desktop, click on the `Start` button and select `Programs -> Accessories -> Hyperterminal`.

Step 3. Double-click the `Hyper Terminal` icon and enter a name for your Hyperterminal session.

Step 4. Select an icon for the new Hyperterminal session. A new icon is created with the name of your session associated with it. You can select this icon the next time you establish a Hyperterminal session.

Step 5. Click `OK`.

Step 6. From the **Phone Number** dialog, select `Connect Using` and then select the communications port to be used to connect to the target system. Click `OK`.

Step 7. In the `Port Settings` tab, enter the following settings:

```
Bits per second = 9600

Data Bits = 8

Parity = None

Stop bits = 1

Flow control = XOn/XOff
```

Step 8. Click OK.

Step 9. From the Hyperterminal window, select Call -> Connect from the pull-down menu to establish your terminal session with the target board. When you are connected, the bottom left of your Hyperterminal screen displays *connected*.

Step 10. Turn on the target system. A power-on banner and PPC1-Bug> prompt should appear on the display terminal.

Note

If your target system already has an OS-9 ROM image installed, you can get a PPC1-Bug> prompt by pressing the Esc key during the target system bootup. You can then rebuild the ROM image as desired.

Building the OS-9 ROM Image with the Configuration Wizard

For More Information

For more information on the OS-9 ROM image and the Configuration Wizard, refer to the ***Getting Started with OS-9*** manual.

The Motorola CompactPCI® reference boards enable you to boot from a number of devices, including the following devices:

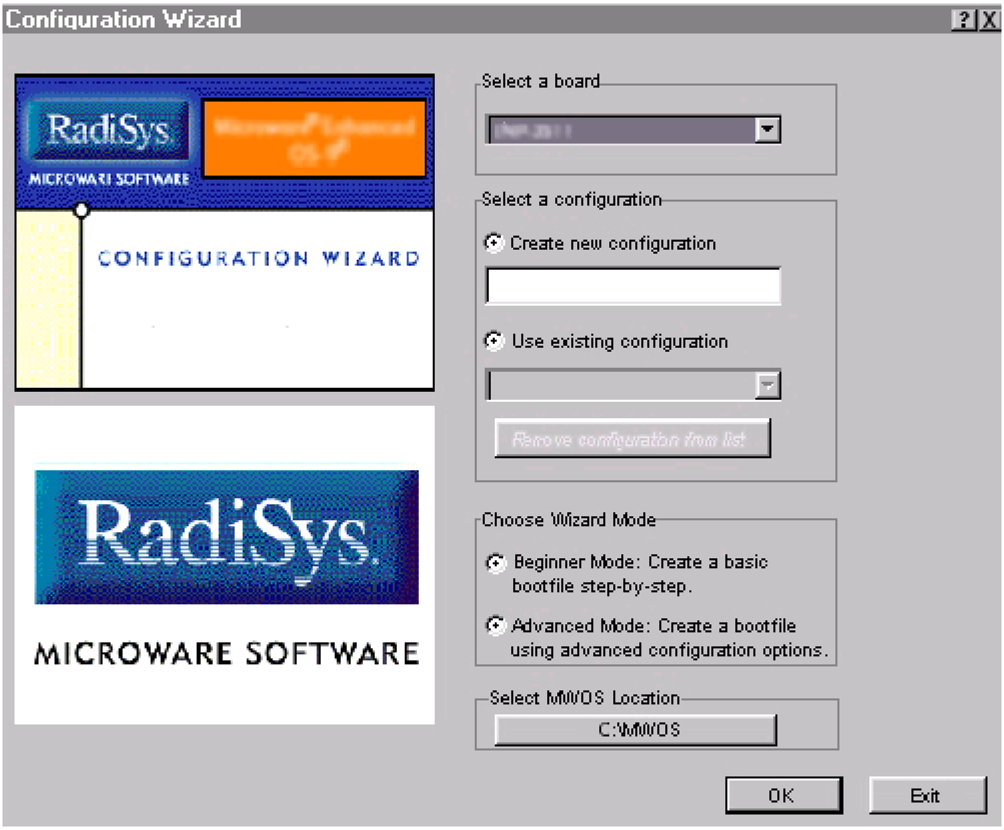
- Flash ROM
- RAM disk
- IDE hard disk
- floppy disk
- Ethernet (you will have to supply your own BOOTP server)

Starting the Configuration Wizard

The Configuration Wizard is the application used to build the coreboot, bootfile, or ROM image. To start the Configuration Wizard, perform the following steps:

Step 1. From the Windows desktop, select `Start` -> `RadiSys` -> `Microware OS-9 for <product>` -> `Configuration Wizard`. You should see the following opening screen:

Figure 1-2 Configuration Wizard Opening Screen

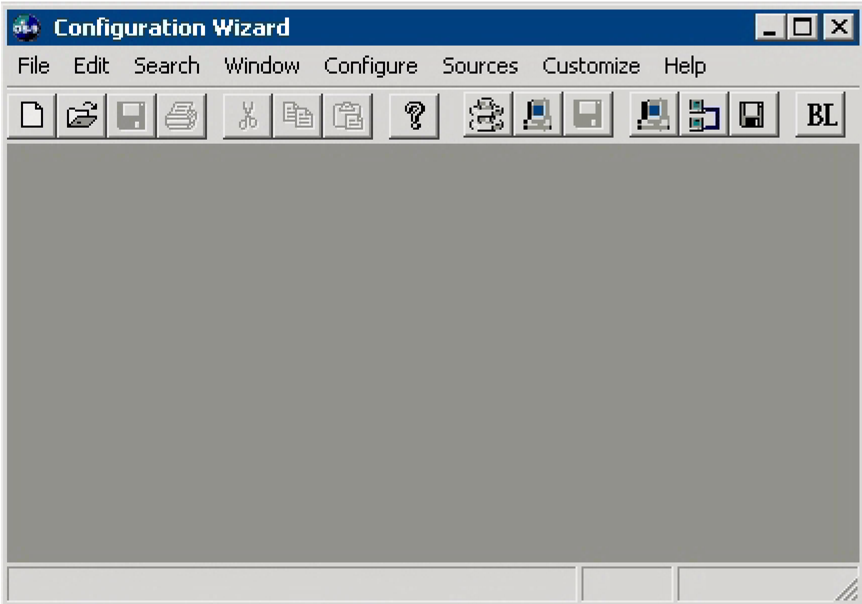


Step 2. Select your target board from the **Select a board** pull-down menu.

Step 3. Select the `Create new configuration` radio button from the **Select a configuration** menu and type in the name you want to give your ROM image in the supplied text box. This names your new configuration, which can later be accessed by selecting the **Use existing configuration** pull down menu.

Step 4. Select the Advanced Mode radio button from the **Choose Wizard Mode** field and click OK. The Wizard's main window is displayed. This is the dialog from which you will proceed to build your image. An example is shown in **Figure 1-3**.

Figure 1-3 Configuration Wizard Main Window



Creating and Configuring the ROM Image

The ROM image consists of the coreboot image and the bootfile image. Together these files comprise the OS-9 operating system.

The Configuration Wizard enables you to choose the contents of your OS-9 implementation. It also enables you to create individual coreboot and bootfile images, or combine them into a single file (the ROM image). The following sections describe how to use the Configuration Wizard to create and configure your OS-9 ROM image.

Creating the Bootfile Image

The default settings in the Configuration Wizard have been preset for optimum performance for the MCP750. The only modifications required are to enable networking and to change the network settings. The network settings information must be obtained from your network administrator.

Note

This section provides an example of an OS-9 ROM image successfully built on a Host PC and transferred to an MPC750 Compact PCI target board. You may have to modify your selections depending on your application.

Select System Type

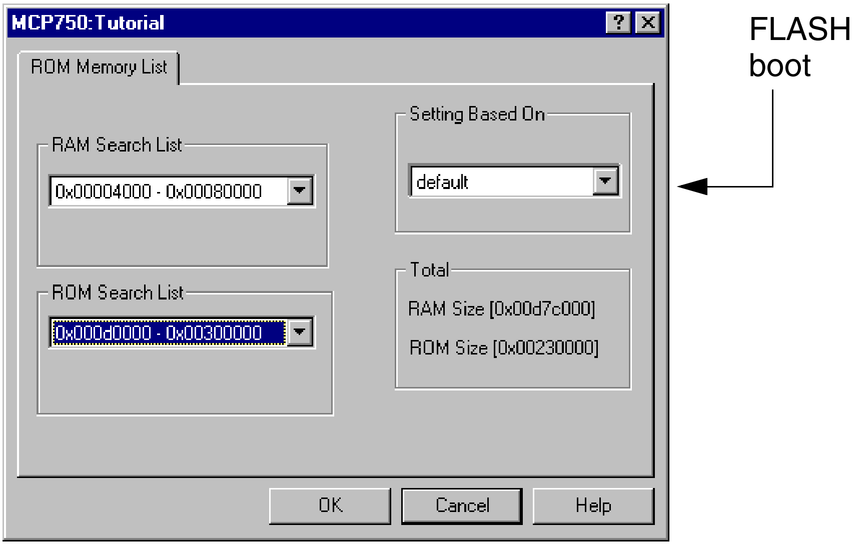
Configure system type options by selecting `Configure -> Sys -> Select System Type` from the **Main Configuration** window.

Note

For the Compact PCI target board, set up the system to boot from Flash.

Configure Flash booting options by selecting the **Flash Boot** option in the **ROM Memory List** tab. The FLASH boot option is in the **Settings Based On** section of the window. Figure **Figure 1-4** shows this configuration.

Figure 1-4 ROM Memory List



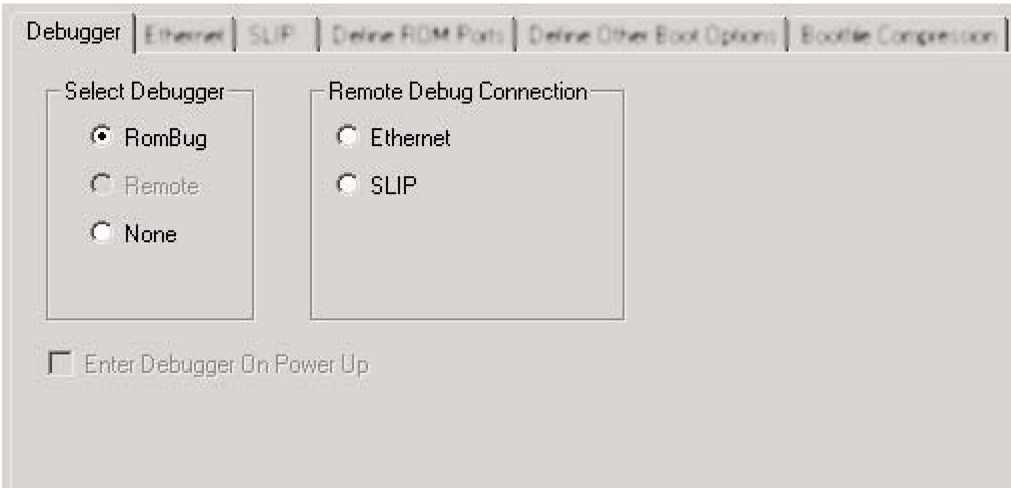
Configure Coreboot Options

To configure your coreboot options, complete the following steps:

Step 1. From the **Main Configuration** window, select `Configure` -> `Coreboot` -> `Main configuration`.

Step 2. Select the **Debugger** tab. The window shown in **Figure 1-5** is displayed.

Figure 1-5 Coreboot Configuration—Debugger Tab



Step 3. Under **Select Debugger**, select `RomBug`. This sets Ethernet as the method for user state debugging. Select `None` if you do not want to debug your program.

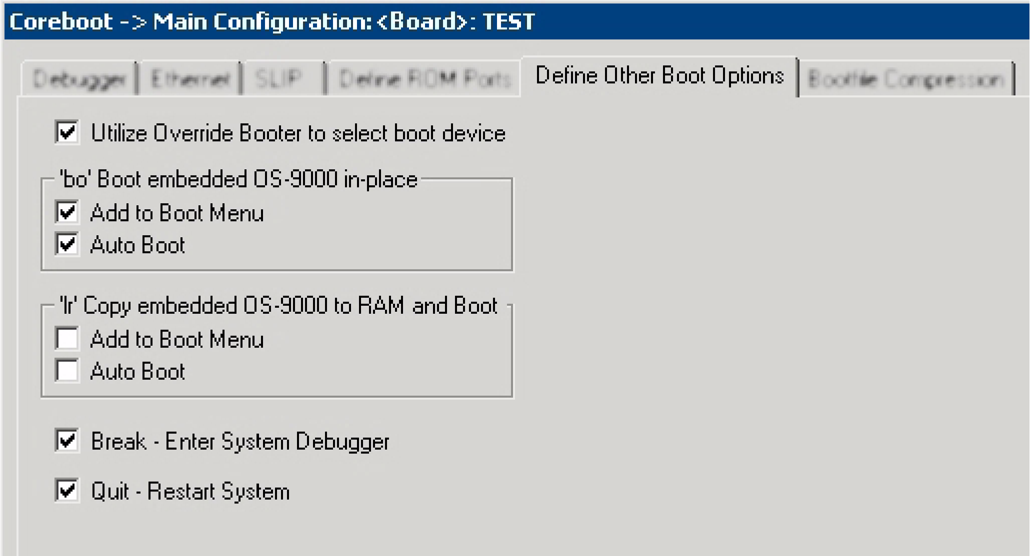
Note

To perform system state debugging, select `Ethernet` under **Remote Debug Connection**. If you set Ethernet as the method for system state debugging, you will not be able to perform user state debugging via Ethernet.

For system state debugging, you must also set the parameters in the **Ethernet** tab of the coreboot configuration.

Step 4. Select the **Define Other Boot Options** tab. The window shown in **Figure 1-6** is displayed.

Figure 1-6 Coreboot Configuration—Define Other Boot Options Tab



Step 5. Select `Break-Enter System Debugger`.

Step 6. Set the `lr` option. The `lr` option moves the boot image modules to RAM before booting. This is optional but since the Flash device is very slow this is highly recommended.

Step 7. Click `OK` and return to the **Main Configuration** window.

Configure System Options

When you select `Configure -> Bootfile -> Configure System Options` the **System Options** window appears. This window contains the **Define /term Port** tab and the **Bootfile Options** tab. Use the default settings for your selections.

Network Configuration

To use the target board across a network, complete the following steps:

Step 1. Select `Configure -> Bootfile -> Network Configuration` from the Wizard's main menu.

Step 2. From the **Network Configuration** dialog, select the `Interface Configuration` tab. From here you can select and enable the interface. For example, you can select the appropriate Ethernet card from the list of options on the left and specify whether you would like to enable IPv4 or IPv6 addressing.

For More Information

To learn more about IPv4 and IPv6 functionalities, refer to the ***Using LAN Communications*** manual, included with this product CD.

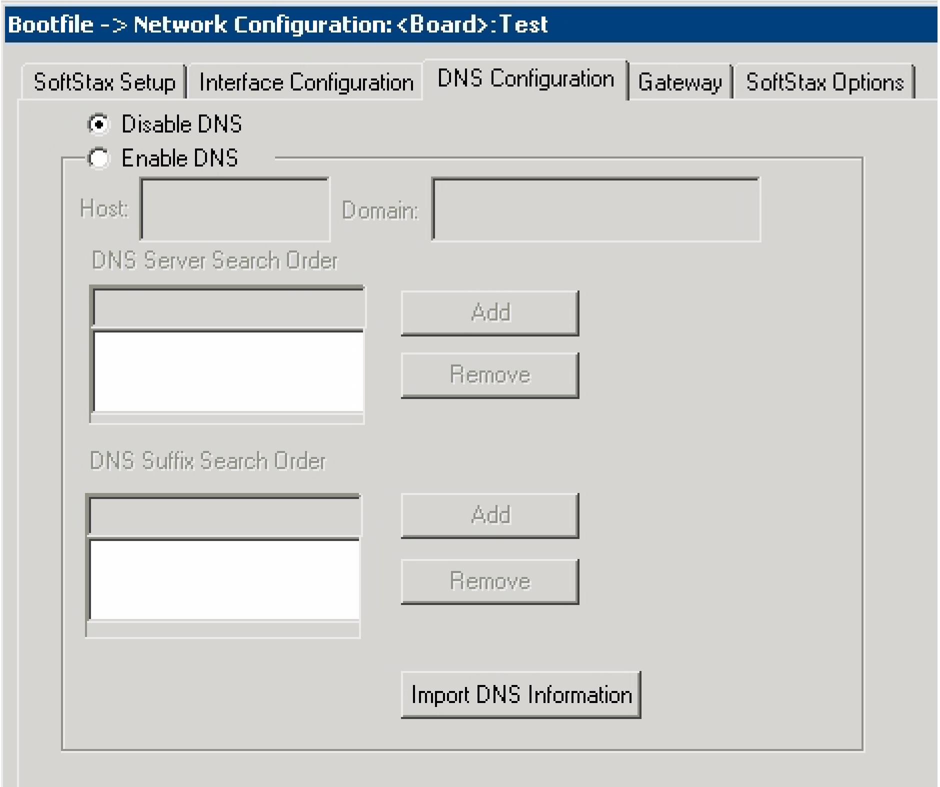
For More Information

Contact your system administrator if you do not know the network values for your board.

Step 3. Once you have made your settings in the **Network Configuration** dialog, click `OK`.

Step 4. Select the **DNS Configuration** tab. The window shown in **Figure 1-7** is displayed. More than one DNS server can be added in this dialog box.

Figure 1-7 Bootfile Configuration—DNS Configuration Tab



If your network does not use DNS, click `Disable DNS`, and move to the Gateway tab.

If you have DNS available, click `Enable DNS` and type your host name and domain.

Note

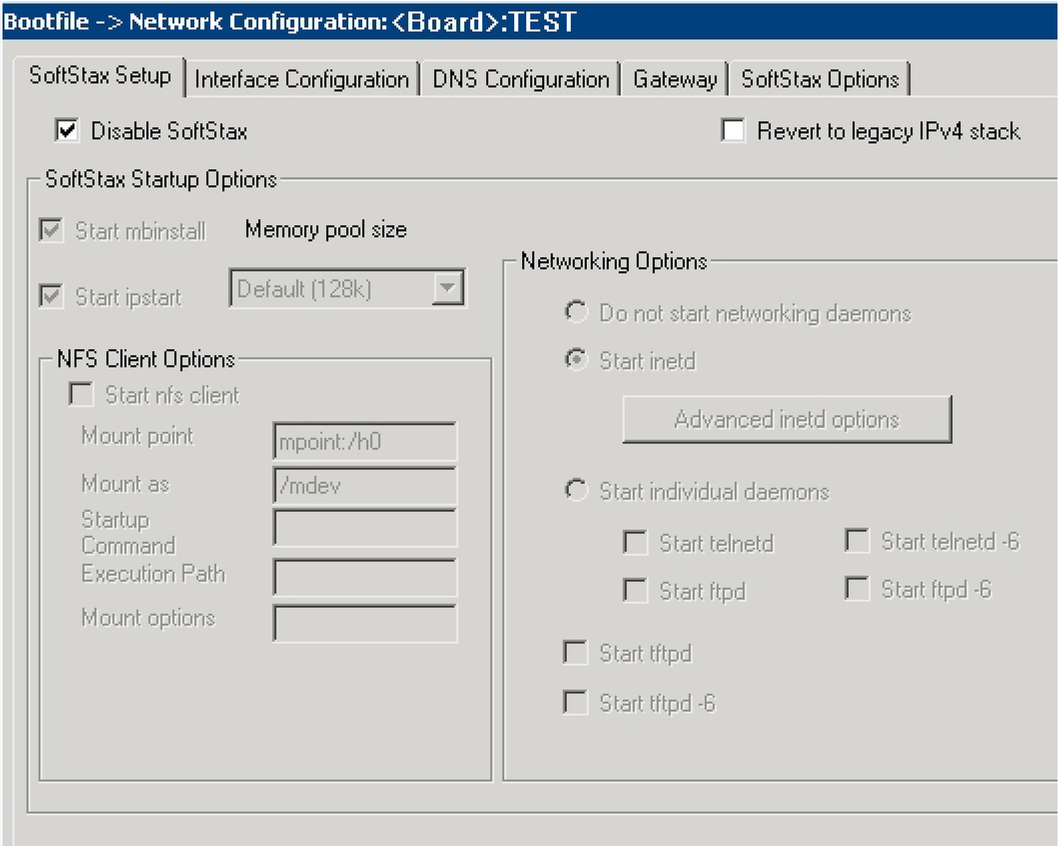
Add DNS IP addresses by clicking on the box directly under **DNS Server Search Order** and typing the IP address. Click the `Add` button when complete. More than one DNS server can be added by repeating these steps.

Step 5. Select the **Gateway** tab. Add new gateway addresses by clicking on the box and typing in the gateway name. Click the `Add` button when complete.

Step 6. Select the `SoftStax`® `Setup` tab. The window shown in **Figure 1-8** is displayed.

The options below represent daemons that can be automatically started if you want to FTP or telnet from a PC to the OS-9 target. **Start NFS Client** enables you to remote mount the target. For this demonstration, you will telnet to the target and establish a sender window and a receiver window.

Figure 1-8 Bootfile Configuration—SoftStax Setup Tab



Step 7. Click `Enable SoftStax`.

Step 8. Click `Start inetd`.

Step 9. Click `OK`.

Step 10. Select the `SoftStax Options` tab.

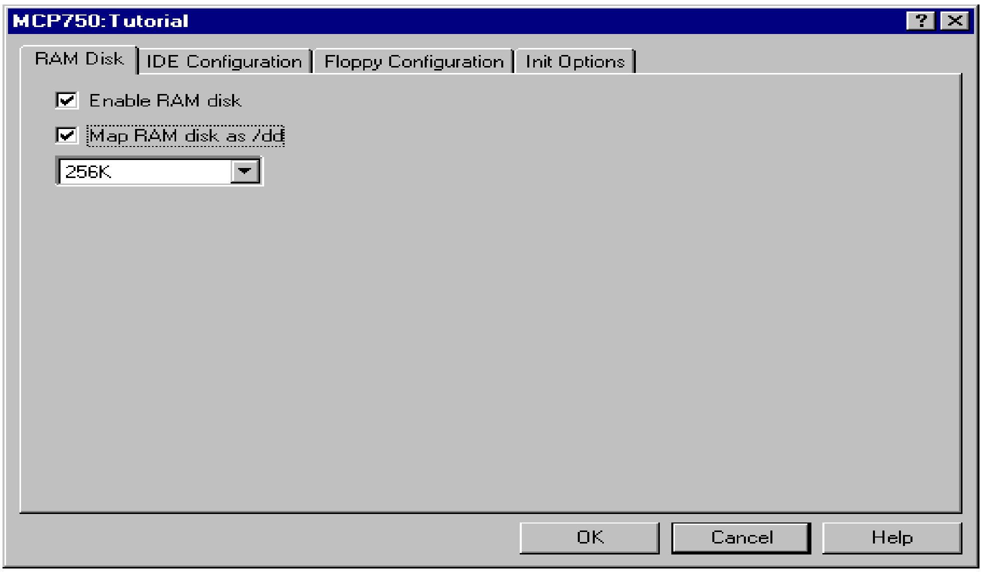
The **SoftStax Options** tab enables you to include networking utilities in the ROM image. By default, `ftp`, `hostname`, `ping`, and `netstat` are included. You can add other utilities as desired.

Step 11. Click `OK` at the bottom of the `Network Configuration` menu to complete network configuration and return to the **Main Configuration** window.

Disk Configuration

Step 1. From the main configuration window, select `Configure -> Bootfile -> Disk Configuration`. The window shown in **Figure 1-9** is displayed.

Figure 1-9 Bootfile Configuration—Disk Configuration Interface



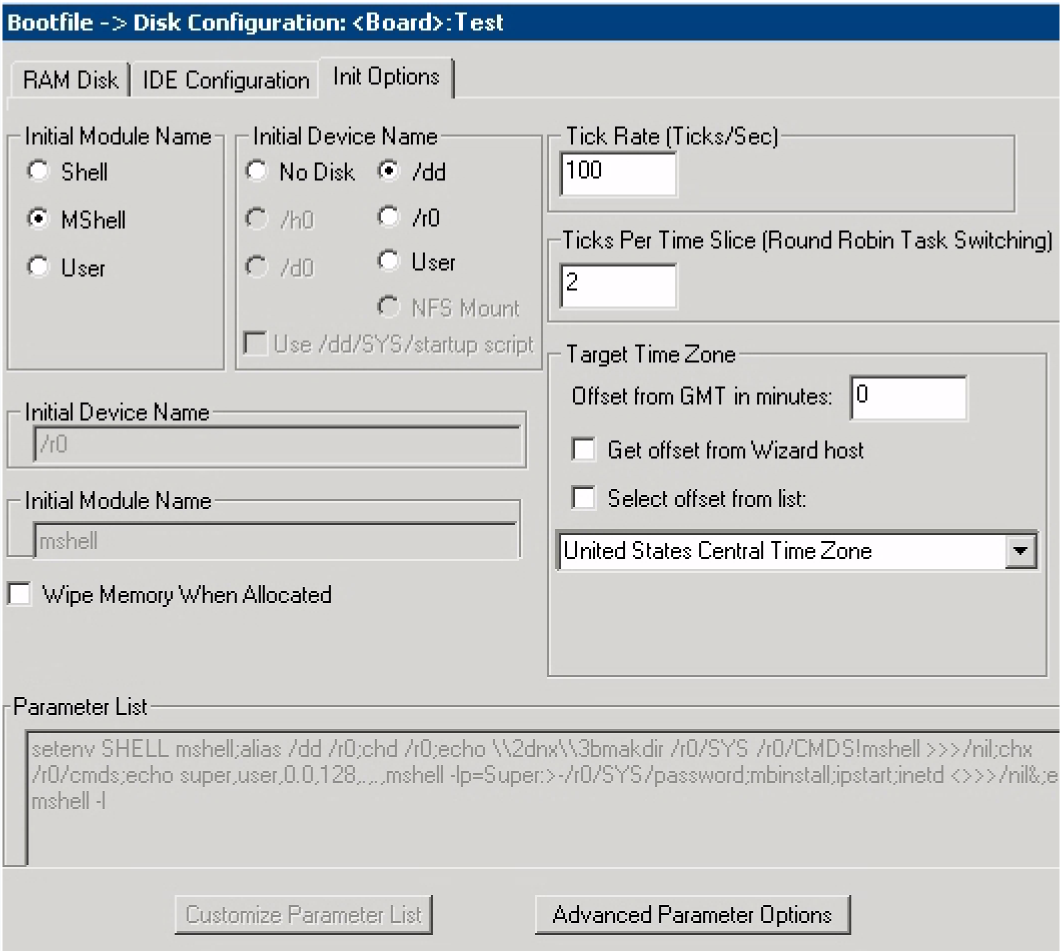
The **Disk Configuration** window contains the following tabs:

- The **RAM Disk** tab enables you to create a RAM disk of any size for loading modules onto the target.

- The **SCSI Configuration** tab enables you to configure SCSI drives for the target.
- The **Floppy Configuration** tab enables you to configure a floppy drive for the target.
- The **Init Options** tab sets the configuration for OS-9 to initialize itself on the target.

Step 2. Select the `Init` tab. The window shown in **Figure 1-10** is displayed.

Figure 1-10 Bootfile Configuration—Init Options Tab



- Select the `Mshell` option for the initial module name. This causes OS-9 to start a console shell usable from your terminal window. Select `No Disk` in the **Initial Device Name** section.
- The tick rate is `100` and ticks per timeslice is set to `2`. If you look at the **Parameter** list box, you can see the commands that OS-9 executes upon system start-up.

Step 3. Click `OK` to return to the **Main Configuration** window.

Build Image

Complete the following steps to build the target board image.

Step 1. From the Main Configuration window, select `Configure -> Build Image`. The **Master Builder** window appears.

Step 2. Select `Coreboot + Bootfile`, the `ROM Utility Set`, the `User State Debugging Modules`, and the `SoftStax (SPF) Support` box under the Include options.

Step 3. Click `Build`. This should display progress information and show the statistics of the image just created.

Step 4. Click `Save As`. The rom file is created in the following directory:

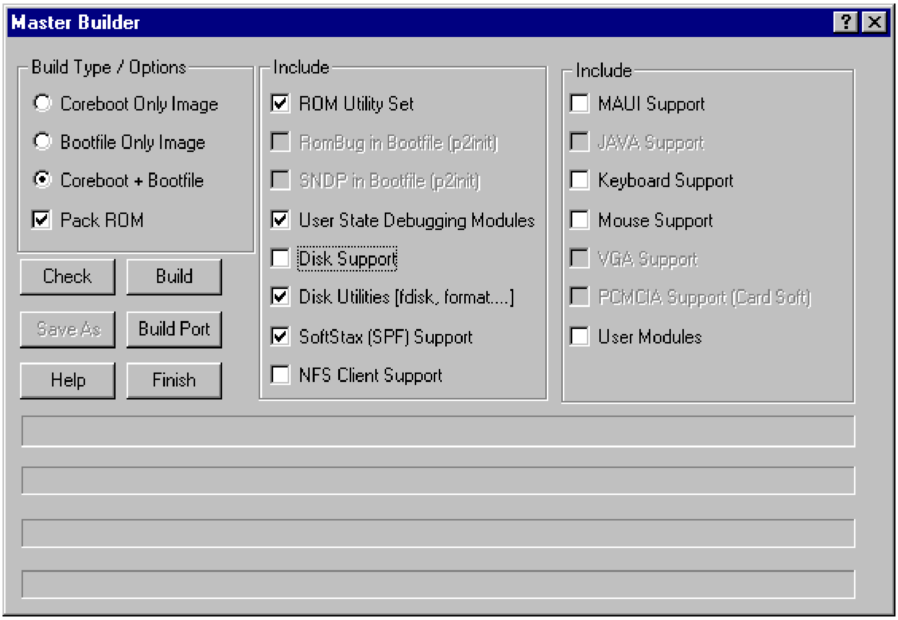
`MWOS/OS9000/603/PORTS/MOTRAVEN/BOOTS/INSTALL/PORTBOOT.`

The rom file can be copied to another name and/or location.

Step 5. Click `Save`. The **Master Builder** window is displayed.

Figure 1-11 shows the **Master Builder** window configuration.

Figure 1-11 Master Builder Window



At this point you can either close the Configuration Wizard or leave it open for use in the section. If you choose to close, you can save your configuration settings for later use.

Transfer the ROM Image to the Target

Complete the following steps to transfer the ROM image to the target board.

Step 1. Start and configure **TFTP SERVER PRO**.

`TFTPServer32` is the Trivial File Transfer Protocol (TFTP) server utility installed on your PC host from the OS-9 for PowerPC CD. This is the tool you will use to transfer the ROM image to the reference board.

To start `TFTPServer32`, click the `Start` button on the Windows desktop.

Select `Programs` --> `TFTPServer` --> `TFTPServer32`.

In the TFTP application, go to the menu and select `System` --> `Setup` and click the `Outbound` tab. The path to where the ROM image is located is shown in the **Outbound File Path** box.

Figure 1-12 TFTP Server Options Window

All other tab settings use the default settings.

Click `OK` to apply the changes and exit.

Step 2. At the `PPC1-Bug>` prompt, enter the Network I/O Physical command:

```
 niot
<return>
```

Note

The `NIOT` command enables you to get files from the supported Ethernet network interfaces and put files to the supported Ethernet network interfaces. When invoked, this command goes into an interactive mode, prompting you for all parameters necessary to carry out the command. This command uses the TFTP protocol to perform the file transfer.

Note

The transfer can take a minute or more depending on your network conditions. If you are using `TFTPServer32`, you will see a log entry reporting a successful transfer. If the utility appears to be hung or showing no progress, verify that your server IP address is correct.

Step 3. Configure the board to receive the file as follows:

`PPC1-Bug>niot`	
`Controller LUN =00?`	
`Device LUN =00?`	
`Node Control Memory Address =00FA0000?`	should not need to change this
`Client IP Address =182.52.109.68?`	fill in as required
`Server IP Address =182.52.109.53?`	fill in as required
`Subnet IP Address Mask =255.255.255.0?`	fill in as required
`Broadcast IP Address =255.255.255.255?`	fill in as required
`Gateway IP Address =0.0.0.0?`	fill in as required
`Boot File Name ("NULL" for None) =rom?`	name of image to load in tftpboot direcoty
`Argument File Name ("NULL" for None) =?`	
`Boot File Load Address =00080000?`	load address; must be 0x80000
`Boot File Execution Address =00080000?`	execution address; must be 0x80000
`Boot File Execution Delay =00000000?`	no delay required
`Boot File Length =00000000?`	get length automatically
`Boot File Byte Offset =00000000?`	
`BOOTP/RARP Request Retry =00?`	
`TFTP/ARP Request Retry =00?`	
`Trace Character Buffer Address =00000000?`	
`BOOTP/RARP Request Control: Always/When-Needed (A/W)=W?`	
`BOOTP/RARP Reply Update Control: Yes/No (Y/N) =Y?`	
`Update Non-volatile RAM (Y/N)`	

Booting the Target from Flash

WARNING

Follow the steps below carefully. During this procedure, it is possible to overwrite the manufacturer's original Flash image. In this event, you will be required to return the hardware to the manufacturer.

Step 1. From PPC1Bug use the `niop` command to load the image.

```
PPC1-Bug>niop
Controller LUN =00?
Device LUN     =00?
Get/Put        =G?
File Name      =? rom
Memory Address =00004000?
Length         =00000000?
Byte Offset    =00000000?

Bytes Received =&1909180, Bytes Loaded =&1909180
Bytes/Second   =&190918, Elapsed Time =10 Second(s)
PPC1-Bug>
```

Step 2. Use the `pflash` utility built into PPC1Bug to program the image into FLASH.

WARNING

Make sure the jumper settings for your board are correct. The memory at 0xff000000 must be the 4MB or 8MB FLASH image not the 1MB image where PPC1Bug is located. Failure to set up the board correctly can cause the PPC1bug image to be erased resulting in a non-working board.

Step 3. Adjust the number of bytes received to a block boundary.

```
PPC1-Bug> pflash 4000:1D21F0 ff000000;b

 PPC1-Bug>pflash 4000:1D21F0 ff000000;b
Source Starting/Ending Addresses      =00004000/001D61EF
Destination Starting/Ending Addresses =FF000000/FF1D21EF
Number of Effective Bytes             =001D21F0 (&1909232)

Program FLASH Memory (Y/N)? y
```

Note

If the last two digits in HEX are less than 0xf0, change them to 0xf0. If the last two digits are greater than 0xf0, add 100_{16} to that number and change the last two digits to 0xf0. Following is an example:

```
&1909180 = 0x1D21BC
round = 0x1D210xf0
```

The image should now be in the 0XFF000260 section.

Step 4. Use the `env` command to tell PPC1Bug where the image is located.

```
PPC1Bug> env
ROM Boot Enable [Y/N]              = Y?
ROM Boot at power-up only [Y/N]    = N?
ROM Boot Abort Delay               = 1?
ROM Boot Direct Starting Address = FF000278?
ROM Boot Direct Ending Address     = FF000278?
```

The above sequence will set up the system to autoboot using the ROM image. You may also use the `rb` command from PPC1bug to boot the ROM system.

Note

The coreboot image can be placed in FLASH without the bootfile image. This can be desirable if disk booting or eb BOOTP booting.

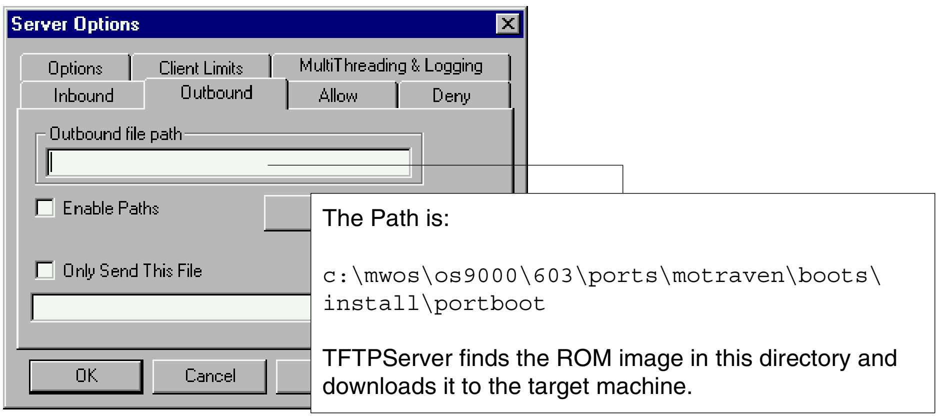
Creating a Startup File

When the Configuration Wizard is set to use a hard drive, or another fixed drive such as a PC Flash Card, as the default device, it automatically sets up the init module to call the `startup` file in the `SYS` directory in the target (For example: `/h0/SYS/startup`, `/mhc1/SYS/startup`). However, this directory and file will not exist until you create it. To create the startup file, complete the following steps:

Step 1. Create a `SYS` directory on the target machine where the `startup` file will reside (for example: `makdir /h0/SYS`, `makdir /dd/SYS`).

Step 2. On the host machine, navigate to the following directory:

```
MWOS/OS9000/SRC/SYS
```



In this directory, you will see several files. The files related to this section are listed below:

- `motd`: Message of the day file
- `password`: User/password file
- `termcap`: Terminal description file
- `startup`: Startup file

Step 3. Transfer all files to the newly created `SYS` directory on the target machine. (You can use Kermit, or FTP in ASCII mode to transfer these files.)

Step 4. Since the files are still in DOS format, you will be required to convert them into the OS-9 format with the `cudo` utility. The following command is an example:

```
cudo -cdo password
```

This will convert the `password` file from DOS to OS-9 format.

For More Information

For a complete description of all the `cudo` command options, refer to the ***Utilities Reference Manual*** located on the Microware OS-9 CD.

Step 5. Since the command lines in the startup file are system-dependent, it may be necessary to modify this file to fit your system configuration. It is recommended that you modify the file before transferring it to the target machine.

Example Startup File

Below is the example startup file as it appears in the `MWOS/OS9000/SRC/SYS` directory:

```
-tnxnp
tmode -w=1 nopause
*
* OS-9 - Version 4.2
* Copyright 2003 by RadiSys Corporation
*
* The commands in this file are highly system dependent and should
* be modified by the user.
*
```

```
setime -s                      ;* start system clock
link mshell csl                ;* make "mshell" and "csl" stay in memory
* iniz r0 h0 d0 t1 p1 term     ;* initialize devices
* load utils                   ;* make some utilities stay in memory
* tsmon /term /t1 &            ;* start other terminals
list sys/motd
setenv TERM vt100
tmode -w=1 pause
mshell<>>>/term -l&
```

For More Information

Refer to the ***Getting Started with OS-9*** manual for more information on startup files.

Optional Procedures

Preliminary Testing

Once you have established an OS-9 prompt on your target system, you can perform the following steps to test your system:

Step 1. Type `mdir` at the prompt.

`mdir` displays all the modules in memory.

Step 2. Type `procs` at the prompt.

`procs` displays the processes currently running in the system.

Step 3. Test the networking on your system.

Select a host on the Ethernet network and run the `ping` utility. The following display shows a successful `ping` to a machine called solkanar.

```
$ ping solkanar
PING solkanar.microware.com (172.16.2.51): 56 data bytes
64 bytes from 172.16.2.51: ttl=128 time=0 ms
```

Step 4. Test `telnet`.

Select a host machine that allows telnet access and try the OS-9 `telnet` utility. The following display shows a successful `telnet` to a machine called delta.

```
$ telnet delta
Trying 172.16.1.40...Connected to delta.microware.com.
Escape character is '^]'.
capture closed.

OS-9/68K V3.0.3 Delta VME177 - 68060 98/12/24 14:41:51
User name?: curt
Password:
Process #101 logged on 98/12/24 14:41:56
Welcome!
***********************************************************
*      WELCOME TO DELTA - THE :OS-9 68K: MACHINE *
```

Step 5. Test telnet from your host PC to the reference board.

From the Windows Start menu, select `Run` and type `telnet` <`hostname`> and click `OK`. A telnet window should display with a `$` prompt. Type `mdir` from the prompt. You should see the same module listing as on the serial console port.

You have now created your OS-9 boot image and established network connectivity with your OS-9 target system.

Booting Your Reference Board from an Ethernet Network

The MCP750 has built-in Ethernet capability. Use the following procedure to set up the board to work on an Ethernet network.

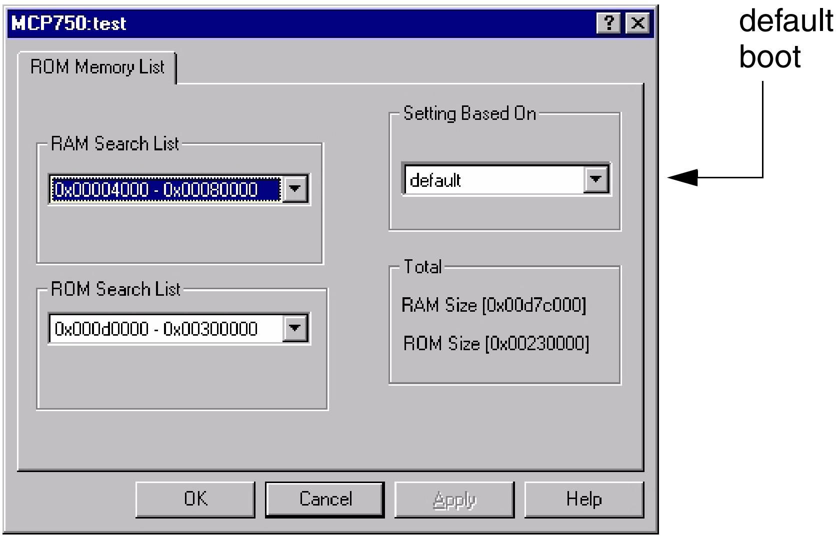
Note

The TFTP SERVER PRO package must be installed from the Microware OS-9 install program. You can also use your own TFTP SERVER.

Step 1. Configure RAM booting options.

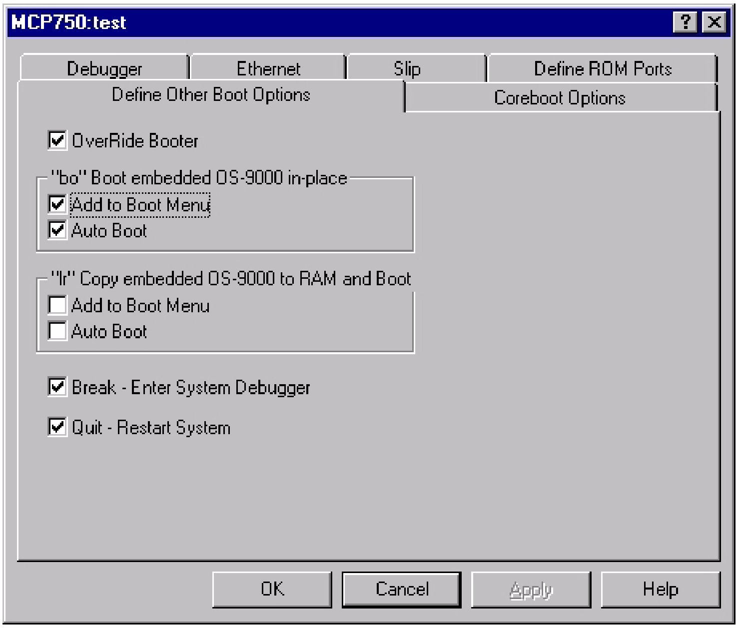
To create an image suitable for Ethernet booting, make sure the default option is selected in the **ROM Memory List** Tab. View this by selecting `Select System Type` from the **Configure** pull-down menu. The default boot option is in the **Settings Based On** section of the window. **Figure 1-13** shows this configuration.

Figure 1-13 ROM Memory List



Also be sure you use the `bo` option. Figure 1-14 shows this configuration. View this screen by selecting `Coreboot -> Main Configuration` from the **Configure** pulldown menu.

Figure 1-14 Setting the `bo` Option



Step 2. Create a ROM image from the build screen and save the image to the `tftpboot` directory.

The ROM image is saved to the following directory on your host system:

`<DRIVE>:\MWOS\OS9000\603\PORTS\MOTRAVEN\BOOTS\INSTALL\PORTBOOT.`

This is the location you browse to in the TFTP server program.

Step 3. Start and configure TFTP SERVER PRO.

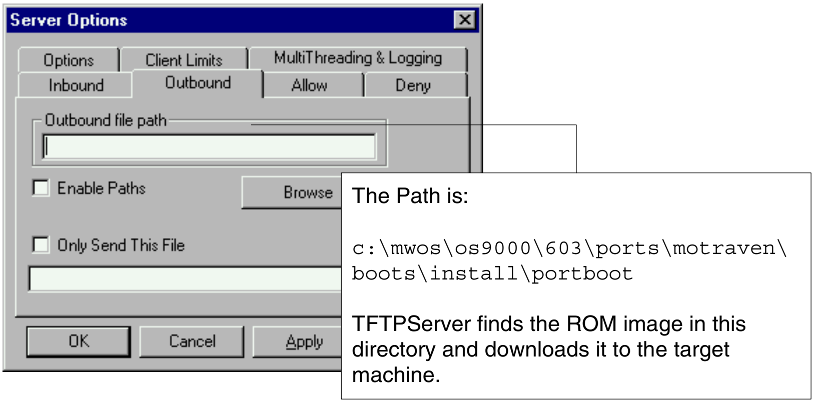
`TFTPServer32` is the Trivial File Transfer Protocol (TFTP) server utility installed on your PC host from the Microware OS-9 for PowerPC CD. This is the tool you will use to transfer the ROM image to the reference board.

To start `TFTPServer32`, click the `Start` button on the Windows desktop.

Select `Programs` --> `TFTPServer` --> `TFTPServer32`.

In the TFTP application, go to the menu and select System --> Setup and click the Outbound tab. The path to where the ROM image is located is shown in the "Outbound File Path" box.

Figure 1-15 TFTP Server Options Window



All other tab settings use the default settings.

Click OK to apply the changes, and exit.

Step 4. At the `PPC1-Bug>` prompt, enter the Network I/O Physical command:

```
niot
<return>
```

Note

The `NIOT` command enables you to get files from the supported Ethernet network interfaces and put files to the supported Ethernet network interfaces. When invoked, this command goes into an interactive mode, prompting you for all parameters necessary to carry out the command. This command uses the TFTP protocol to perform the file transfer.

Note

The transfer can take a minute or more depending on your network conditions. If you are using `TFTPServer32`, you will see a log entry reporting a successful transfer. If the utility appears to be hung or showing no progress, verify that your server IP address is correct.

Step 5. Configure the board to receive the file as follows:

`PPC1-Bug>niot`	
`Controller LUN =00?`	
`Device LUN     =00?`	
`Node Control Memory Address =00FA0000?`	should not need to change this
`Client IP Address =182.52.109.68?`	IP address of the reference board
`Server IP Address =182.52.109.53?`	IP address of the machine with tftp boot server
`Subnet IP Address Mask =255.255.255.0?`	fill in as required
`Broadcast IP Address =255.255.255.255?`	fill in as required
`Gateway IP Address =0.0.0.0?`	fill in as required
`Boot File Name ("NULL" for None) =rom?`	name of image to load in tftpboot direcoty
`Argument File Name ("NULL" for None) =?`	
`Boot File Load Address =00080000?`	load address; must be 0x80000
`Boot File Execution Address =00080000?`	execution address; must be 0x80000
`Boot File Execution Delay =00000000?`	no delay required
`Boot File Length =00000000?`	get length automatically
`Boot File Byte Offset =00000000?`	
`BOOTP/RARP Request Retry =00?`	
`TFTP/ARP Request Retry =00?`	
`Trace Character Buffer Address =00000000?`	
`BOOTP/RARP Request Control: Always/When-Needed (A/W)=W?`	
`BOOTP/RARP Reply Update Control: Yes/No (Y/N) =Y?`	
`Update Non-volatile RAM (Y/N)`	

Step 6. Boot the system by entering `nbo`. Your screen should display the following:

```
PPC1-Bug>nbo
Network Booting from: DEC21040, Controller 0, Device 0
Loading: rom

Client IP Address      = 182.52.109.68
Server IP Address      = 182.52.109.53
Gateway IP Address     = 0.0.0.0
Subnet IP Address Mask = 255.255.255.0
Boot File Name         = rom
Argument File Name     =

Network Boot File load in progress... To abort hit <BREAK>
Bytes Received =&1909180, Bytes Loaded =&1909180
Bytes/Second   =&56152, Elapsed Time =34 Second(s)


OS-9000 Bootstrap for the PowerPC(tm)

Now trying to Override autobooters.
Now trying to Scan SCSI devices.
Symbios 53C810 @ 0x81000000 SELFID (07) MAXCNT (0x01000000)

ID   Vendor   Product          Rev   Block Size  Total Blks  Disk Size
-----------------------------------------------------------------------
0x00 SEAGATE  ST11200N ST31230 0456  0x00000200  0x001f9563  1059768320
0x01 MICROP   4221-09MZ  Q4D   HT02  0x00000200  0x003d197a  2050(MEG)

Now trying to Boot embedded OS-9000 in-place.
Now searching memory ($000d1840 - $002521bf) for an OS-9000 Kernel...

An OS-9000 kernel was found at $000d1840
A valid OS-9000 bootfile was found.
+3
$
```

Note

Use the `env` command to setup the `nbo` option as an autobooter if desired.

```
PPC1Bug> env
Network Auto Boot Enable [Y/N]               = Y?
Network Auto Boot at power-up only [Y/N]     = Y?
Network Auto Boot Controller LUN             = 00?
Network Auto Boot Device LUN                 = 00?
Network Auto Boot Abort Delay                = 5?
Network Auto Boot Configuration Parameters Offset (NVRAM) = 00001000?
```

Chapter 2: Board Specific Reference

This chapter contains information that is specific to the Motorola CompactPCI® reference boards. It contains the following sections:

- **Boot Options**
- **Port Specific Utilities**
- **PowerPC™ Registers Passed to a New Process**

For More Information

For general information on porting OS-9, see the ***OS-9 Porting Guide***.

Boot Options

You select your boot device menu options using the Configuration Wizard. For each boot device option, you can select whether you want it to be displayed on a boot menu, set up to autoboot, or both. The autoboot option enables the device selected to automatically boot up the high-level bootfile, bypassing the boot device menu.

Note

When using the Configuration Wizard, you should select only one device for autoboot on your system.

Following is an example of the Boot Menu displayed in the terminal emulation window (using Hyperterminal):

```
OS-9000 Bootstrap for the PowerPC(tm)

Now trying to Override autobooters.

BOOTING PROCEDURES AVAILABLE ----- <INPUT>

Boot FDC floppy ------------------ <fd>
Boot from PC-Floppy -------------- <pf>
Boot from SCSI PC-Floppy --------- <pfs>
Boot over Ethernet --------------- <eb>
Boot embedded OS-9000 in-place --- <bo>
Enter system debugger ------------ <break>
Restart the System --------------- <q>

Select a boot method from the above menu:
```

What you select for boot options determines what modules are included in the coreboot image. Table 2-1 lists some of the supported boot devices for OS-9.

Table 2-1 Supported Boot Methods

Type of Boot	Description
Floppy Disk	Boot from floppy disk. You must select if the floppy is controlled by a Random Block File System (RBF) (`fd` or `fs`) or PC File System (`pf` or `pfs`).
Boot embedded OS-9 in-place	Boot OS-9 from FLASH (`bo`).
Copy embedded OS-9 to RAM and Boot	Copy OS-9 from FLASH (if stored there) to RAM and boot (`lr`).

Configuring Booters

The following booters are available for the MCP750 Compact PCI target platforms. The abbreviated name and configuration parameters for the booters are listed with recommended values.

Table 2-2 MCP750 Booters

Booter	Description	Recommended Values
`fdc765`	Standard floppy disk booter	
	Abbreviated name:	`"fd"`
	Configuration parameters:	`"port=0x800003f0"`
		`"lun=0"`
		`"si=0"`
		`"ei=3"`
`ide`	Standard IDE hard disk booter	
	Abbreviated name:	`"ide"`
	Configuration parameters:	`"port=0x800001f0"`
		`"si=0"`
		`"ei=3"`
		`"lsnoffs=2052"`

Table 2-2 MCP750 Booters

Booter	Description	Recommended Values
`llbootp`	Standard BOOTP booter	
	Abbreviated name:	`"eb"`
	Configuration parameters	`"driver=l121040"`
`romboot`	Embedded system booter	
	Abbreviated name:	`"ro" (reconfigured to "bo" and "lr")`
	Configuration parameters:	`<none>`

Vector Descriptions for PowerPC 603/604

Table 2-3 Vector Descriptions for PowerPC 603/604

Vector Number	Related OS-9000 Call	Assignment
0	None	Reserved
1	F_IRQ	System reset
2	F_STRAP, F_IRQ	Machine check
3	F_STRAP, F_IRQ	Data access
4	F_STRAP, F_IRQ	Instruction access
5	F_IRQ	External interrupt
6	F_STRAP, F_IRQ	Alignment
7	F_STRAP, F_TLINK, F_IRQ	Program
8	F_IRQ	Floating-point unavailable
9	F_IRQ	Decrementer
10	None	Reserved
11	None	Reserved
12	F_SSVC	System call
13	None	Trace
14	None	Reserved

Table 2-3 Vector Descriptions for PowerPC 603/604

Vector Number	Related OS-9000 Call	Assignment
15	None	Reserved
	F_IRQ	Performance monitoring interrupt (604e)
16	None	Instruction translation miss
	None	Reserved (604e)
17	None	Data load translation miss
	None	Reserved (604e)
18	None	Data store translation miss
	None	Reserved (604e)
19	F_IRQ	Instruction address breakpoint
20	F_IRQ	System management interrupt
21-47	None	Reserved

Note

The vector numbers in **Table 2-3** are logical vector numbers. The actual processor vectors can be computed by multiplying the logical vector number by 256.

Error Exceptions: vectors 2-4 and 6-7

These exceptions are usually considered fatal program errors and unconditionally terminate a user program. If `F_DFORK` creates the process or the process had `debug` attached with `F_DATTACH`, then the resources of the erroneous process remain intact and control returns to the parent debugger to allow a post-mortem examination.

A user process may use the `F_STRAP` system call to install an exception handler to catch the errors and recover from the exceptional condition. When a recoverable exception occurs, the process' exception handler installed with the `F_STRAP` system call is executed with a pointer to the process' normal static data and the current stack pointer. Also, the process' exception handler will receive as parameters the vector number of the error, the program instruction counter of where the error occurred, and the fault address of the error if applicable. The exception handler must decide whether or not to continue execution. Programs written in the C language may use the `setjmp` and `longjmp` library routines to properly recover from the erroneous condition.

If any of these exception occur in system state during a system call made by the process due to the process passing bad data to the kernel, the process' exception handler is not called. Instead, the appropriate vector error is returned from the system call.

Vectored Interrupts: vector 5

In general, the PowerPC processor family uses a single interrupt vector for all external interrupts. However, most systems supporting the PowerPC family use additional external logic to support more powerful nested interrupt facilities. Hence, the vector numbers used by OS-9000 device drivers are usually logical vectors outside of the range of the hardware vectors listed above. The device drivers install their interrupt service routines via the `F_IRQ` system call on the logical vector. The kernel's dispatch code uses the external logic vector to identify the source of the interrupt and to call the associated interrupt service routine. Interrupt service routines are executed in system state without an associated current process.

Note

The F_IRQ system call may also be used to install exception handlers on some non-hardware interrupt vectors. The above table lists the exceptions that may be monitored using the F_IRQ facility. The installed exception handler is called just like any other interrupt service routine when the associated exception occurs.

User Trap Handlers: vector 7

This vector is used for dispatching user code into system state trap handlers. The vector provides a mechanism for programs to switch states and dispatch to a subroutine module, in order to execute code in system state.

System Calls: vector 12

This vector is used for service call dispatching to the OS-9000 operating system. It is also useful. for user services installed using the F_SSVC service request.

PowerPC™ Registers Passed to a New Process

The following PowerPC registers are passed to a new process (all other registers are zero):

```
r1  = stack pointer
r2  = static storage (data area) base pointer
r3  = points to fork parameters structure (listed in
                                           f_fork)
r13 = points to the constant data of code area of the
                                           module
```

Note

`r2` is always biased by the amount specified in the `m_dbias` field of the program module header which allows object programs to access a larger amount of data using indexed addressing. You can usually ignore this bias because the OS-9000 linker automatically adjusts for it.

Port Specific Utilities

The following port specific utilities are included:

- `dmppci`
- `mouse`
- `pciv`
- `setpci`
- `testpci`

SYNTAX

```
dmppci <bus_number> <device_number>
       <function_number> {<size>}
```

OPTIONS

`-?` Display help

DESCRIPTION

`dmppci` displays PCI configuration information that is not normally available by other means, except programming, using the PCI library.

EXAMPLE

```
$ dmppci 0 11 1 0x40
    PCI DUMP Bus:0 Dev:11 Func:1 Size:64
    -----------------------------------
VID  DID  CMD  STAT CLASS  RV CS IL IP LT HT BI MG ML SVID SDID
---  ---- ---- ---- -----  -- -- -- -- -- -- -- -- -- ---- ----
10ad 0105 0005 0280 01018f 05 08 0e 01 00 80 00 02 28 0000 0000

BASE[0]  BASE[1]  BASE[2]  BASE[3]  BASE[4]  BASE[5]  CIS_P    EXROM
-------- -------- -------- -------- -------- -------- -------- --------
01000321 01000331 01000329 01000335 01000301 01000311 00000000 00000000

Offset 00 01 02 03 04 05 06 07 08 09 0a 0b 0c 0d 0e 0f
-----------------------------------------------------
0000   ad 10 05 01 05 00 80 02 05 8f 01 01 08 00 80 00
0010   21 03 00 01 31 03 00 01 29 03 00 01 35 03 00 01
0020   01 03 00 01 11 03 00 01 00 00 00 00 00 00 00 00
0030   00 00 00 00 00 00 00 00 00 00 00 00 0e 01 02 28
```

mouse — Show Mouse Library Functions

SYNTAX

```
mouse &lt;opts&gt;
```

OPTIONS

`-?`	Display help
`-s`	Slow mouse
`-f`	Fast mouse
`-r[n]`	Set resolution to n
`-p[n]`	Set sample rate to n
`-c[n]`	Set scale factor to n

DESCRIPTION

`mouse` displays mouse status information.

EXAMPLE

```
$ mouse
Opening device /m0
status = 0x08, x =   4, y =   0
status = 0x08, x =   6, y =   0
status = 0x08, x =   7, y =   1
status = 0x08, x =   7, y =   1
status = 0x08, x =   8, y =   1
status = 0x08, x =   7, y =   0
status = 0x28, x =   7, y = 255  Y Negative
status = 0x28, x =   7, y = 254  Y Negative
status = 0x28, x =   5, y = 254  Y Negative
status = 0x08, x =   2, y =   0
status = 0x28, x =   1, y = 255  Y Negative
status = 0x08, x =   2, y =   0
status = 0x28, x =   0, y = 255  Y Negative
status = 0x08, x =   1, y =   0
status = 0x09, x =   0, y =   0  Left Button
status = 0x08, x =   0, y =   0
status = 0x0a, x =   0, y =   0  Right Button
status = 0x08, x =   0, y =   0
```

pciv PCI Configuration Space View

SYNTAX

```
pciv [<opts>]
```

OPTIONS

`-?`	Display help.
`-a`	Display base address information and size.
`-r`	Display PCI routing information.

DESCRIPTION

The `pciv` utility allows visual indication of the status of the PCIbus. This utility is port dependent.

EXAMPLES

When using the `pciv` command with a Motorola PowerPC board, the following information (or something similar) is displayed:

```
$ pciv

PowerPC 603 Configuration Report

Model:  Ultra PowerPC

Board Configuration Reports
[Z85230 ESCC] [PMC] [Graphics] [Ethernet] [SCSI]

BUS:DV:FU  VID  DID  CMD  STAT CLASS  RV CS IL IP
------------------------------------------------
000:00:00  1057 0001 0106 2080 060000 24 00 00 00 MPC105
000:11:00  8086 0484 000f 0200 000000 84 00 00 00 PCI/ISA Bridge
000:12:00  1000 0001 0007 0200 010000 02 00 0b 01 NCR53C810 SCSI
000:14:00  1011 0002 0007 0280 020000 23 00 09 01 DECchip 21040
000:15:00  1013 00a8 0000 0000 030000 8e 00 0b 01 GD5434 Graphics
```

The following configuration registers apply to these `DEV` columns:

- 12 - NCR53C810 Configuration Register
- 14 - DECchip 21040 Configuration Register
- 15 - GD5434 Configuration Register

The `pciv` command in the previous example reports configuration information related to specific hardware attached to the system. The MCP750 series is specific about the PCI devices located on the main board. For this reason, the information displayed is not generic in format.

```
DETAIL OF BASIC VIEW:
   BUS        : Bus Number
   DEV        : Device Number
   VID        : Vendor ID
   DID        : Device ID
   CLASS      : Class Code
   RV         : Revision ID
   IL         : Interrupt Line
   IP         : Interrupt Pin
   [S]        : Single function device
   [M]        : Multiple function device
```

When the `-a` option is used address information is also displayed as well as the size of the device blocks being used. All six address PCI address entries are scanned.

```
(C) [32-bit] base_addr[0] = 0x3efefe81  PCI/IO
                  0xbefefe80 Size = 0x00000080
```

The fields in the previous example are, from left to right, as follows:

- Prefetchable
- Memory Type
- Address Fields
- Actual Value Stored
- Type of Access
- Translated Access Address Used (shown on second line)
- Size of Block (shown on second line)

When the `-r` option is used, PCI-specific information related to PCI interrupt routing is displayed. If an ISA BRIDGE controller is found in the system, the routing information is used. The use of ISA devices and PCI devices in the same system requires interrupts to be routed either to ISA or PCI devices. Since ISA devices employ edge-triggered interrupts and PCI use devices use level interrupts, the `EDGE/LEVEL` control information is also displayed. If an interrupt is shown as `LEVEL` with a PCI route associated with it, no ISA card can use that interrupt. This command also shows the system interrupt mask from the interrupt controller.

Note

ISA and PCI interrupts cannot be shared.

SYNTAX

```
setpci <bus> <dev> <func> <offset> <size{bwd}> <value>
```

OPTIONS

`-?`	Display help

DESCRIPTION

The `setpci` utility sets PCI configuration information that is not normally available by other means other than programming using the PCI library. The `setpci` utility may also be used to read a single location in PCI space. Parameters include:

`<bus>`	= PCI Bus Number 0..255
`<dev>`	= PCI Device Number 0..32
`<func>`	= PCI Function Number 0..7
`<offset>`	= Offset value (ie. command register offset = 4)
`<size>`	= Size b=byte w=word d=dword
`<value>`	= The value to write in write mode. If no value is included, the utility is in read mode.

EXAMPLES

```
$ setpci 0 19 0 0x14 d

PCI READ MODE
-------------

PCI Value.....0x3bfedd00 (dword) READ

PCI Bus.........0x00
PCI Device......0x13
PCI Function....0x00
PCI Offset....0x0014

$ setpci 0 19 0 0x14 d 0x1234500

PCI WRITE MODE
--------------

PCI Value.....0x01234500 (dword) WRITE

PCI Bus.........0x00
PCI Device......0x13
PCI Function....0x00
PCI Offset....0x0014
$
$ setpci 0 19 0 0x14 d

PCI READ MODE
-------------

PCI Value.....0x01234500 (dword) READ

PCI Bus.........0x00
PCI Device......0x13
PCI Function....0x00
PCI Offset....0x0014
```

`testpci` Test PCI Value

SYNTAX

```
testpci
```

OPTIONS

`-?` Display help

DESCRIPTION

The `testpci` utility tests all PCI library functions. To use this utility, you must have a graphics card in the system. This utility shows how the PCI library calls can be used.

EXAMPLE

```
$ testpci
Test PCI Library Calls Edition 2
_pci_search_device ......................ok....
_pci_next_device ........................ok....
_pci_get_config_data ....................ok....
_pci_find_device ........................ok....
_pci_find_class_code ....................ok....
_pci_read_configuration_byte ............ok....
_pci_read_configuration_word ............ok....
_pci_read_configuration_dword ...........ok....
_pci_write_configuration_byte ...........ok....
_pci_write_configuration_word ...........ok....
_pci_write_configuration_dword ..........ok....
_pci_get_irq_pin ........................ok....
_pci_get_irq_line .......................ok....
_pci_set_irq_line .......................ok....
PCI LIBRARY TEST CONTAINS NO ERRORS.
```

Appendix A: Board Specific Modules

This chapter contains an overview of the board-specific low-level system modules and the high-level system modules. Each listing includes a brief description. The following sections are included:

- **Low-Level System Modules**
- **High-Level System Modules**
- **Common System Modules List**

For More Information

For a list of all of the OS-9 modules common to all boards, see the ***OS-9 Device Descriptor and Configuration Module Reference***.

Low-Level System Modules

The following low-level system modules are tailored specifically for the Compact PCI target platforms. These modules can be found in the following directory:

`MWOS/OS9000/603/PORTS/MOTRAVEN/CMDS/BOOTOBJS/ROM`

Configuration Modules

`cnfgdata`	provides low-level configuration data including configuration of a serial console.
`cnfgfunc`	retrieves configuration parameters from the `cnfgdata` module.
`commcnfg`	retrieves the name of the low-level auxiliary communication port driver from the `cnfgdata` module.
`conscnfg`	retrieves the name of the low-level console driver from the `cnfgdata` module.

Console Drivers

`io16550`	provides console services for the external `16550` serial ports.
`io8042`	provides console services for the VGA display and keyboard interface (when available).
`io85x30`	provides console services for the `82530` serial ports (when available).

Debugging Module

`usedebug`	is a debugger configuration module.

Ethernet Driver

`l121040`	provides network driver services for the `DEC 21040` Ethernet port.

System Modules

`ide`	is a low-level IDE booter module.
`initext`	is a user-customizable system initialization module.
`portmenu`	retrieves a list of configured booter names from the ROM `cnfgdata` module.
`romcore`	is a bootstrap code.
`rpciv`	shows information about devices on the PCI bus.

Timer Module

`swi8timr`	provides polling timer services with a software loop self-calibrated from the `8259`-like timer.

High-Level System Modules

The following OS-9 system modules are tailored specifically for Compact PCI series platforms. Unless otherwise specified, each module can be found in the following directory:

`MWOS/OS9000/603/PORTS/MOTRAVEN/CMDS/BOOTOBJS`

Interrupt Controllers

These modules provide extensions to the vectors module by mapping the single interrupt generated by an interrupt controller into a range of pseudo vectors which are recognized by OS-9 as extensions to the base CPU exception vectors.

`picirq`	provides interrupt acknowledge and dispatching support for the nested `8259` interrupt controllers on the Compact PCI series platforms. `picirq` also maps the nested PIC interrupts `0-15` to OS-9 pseudo vectors `64-79` (`$40-$4f`). The `picirq` module used in the sample boots is located in the file also called `picirq`. It provides slightly lower performance, but allows use of the last set of BAT registers for ISA memory access. This is the default configuration, as it supports a wider range of platforms.
`universeirq`	provides interrupt acknowledge and dispatch support for the Tundra Universe (`CA91C042`) chip implemented on the Compact PCI series of CPU boards. Use this module together with the proper `picirq` module if you require access to VME interrupts on one of these platforms.

	`universeirq` maps VME interrupts `64-255` to OS-9 pseudo-vectors `64-255` (`$40-$ff`).
`ravenirq`	provides interrupt acknowledge and dispatch support.

Real Time Clock Driver

`rtc48t18`	provides OS-9 access to the M48T18 BBRAM real time clock. In this release, `rtc48t18` is the name of the ticker, regardless of the CPU in use on your platform. This is likely to change in a future release.

Ticker

`tk8253`	provides the system ticker through the Intel 8253 programmable interval timer.

Abort Handler

`abort`	provides a handler for the abort interrupt, which calls into the system-state debugger. If no system state debugger is configured, the system will perform a soft reset.

Shared Libraries

`picsub`	provides interrupt enable and disable routines to handle platform-specific interrupt controller issues for device drivers. This module is called by all drivers and should be included in your `bootfile`.

Serial and Console Drivers

`sc16550`

provides support for the external `16550` serial ports. This driver is used to drive the console over the com1 port in the sample boots provided in the package.

The descriptors provided for this driver are named `t1`, `t2`, `term_t1`, and `term_t2` and are located in the following directory:

```
MWOS/OS9000/603/PORTS/MOTRAVEN/
CMDS/BOOTOBJS/DESC/SC16550
```

`sc85x30`

provides support for the `82530` serial ports (when available). The descriptors provided for this driver are named `t3`, `t4`, `term_3`, and `term_4` and are located in the following directory:

```
MWOS/OS9000/603/PORTS/MOTRAVEN/
CMDS/BOOTOBJS/DESC/SC85X30
```

`sc8042`	provides unified support for the `i8042` keyboard and VGA monitor output device (when available). The descriptors for this device are named `t0` and `term` and are located in the following directory: `MWOS/OS9000/603/PORTS/MOTRAVEN/CMDS/BOOTOBJS/DESC/SC8042` To configure your monitor as the high-level console, change the reference to the `term` device descriptor in the boot list used to build your system to point to this file instead of the `16550` `term` descriptor.
`sc8042k`	provides unified support for the `i8042` keyboard and input device (mouse). The descriptors provided for this driver are named `k0`, `kx`, and `m0` are located in files stored in the following directory: `MWOS/OS9000/603/PORTS/MOTRAVEN>/CMDS/BOOTOBJS/DESC/SC8042K`
`sc8042m`	provides unified support for the multiple windowing version of the SC8042, keyboard, and graphics support in text mode using a standard VGA card and monitor. The descriptors provided for this driver are named `term`, `mterm0`, `mterm1`, `mterm2`, and `mterm3`. For an explanation of the language versions available, see the previous note. The descriptors are located in files stored in the following directory: `MWOS/OS9000/603/PORTS/MOTRAVEN/CMDS/BOOTOBJS/DESC/SC8042M`

Note

For each of the sc8042 keyboard descriptors, several language versions are provided including: French, United Kingdom, German, and Norwegian. The different language descriptors are named according to the same rules as shown in the example for the French i8042 keyboard descriptor: k0_fr.

Parallel Driver

scp87303	provides support for the 87303 parallel port. The descriptor provided for this driver is named p.lp1 and is located in the following directory: MWOS/OS9000/603/PORTS/MOTRAVEN/ CMDS/BOOTOBJS/DESC/SCP87303

Data Disk Drivers

rb765	is a device driver for floppy drive.
rb1003	provides support for IDE and EIDE drives up to 4GB. Many descriptors are provided for use with this driver. Among the descriptors provided are several modules named h0 and dd. These descriptors are contained in files of unique names and located in the following directory: MWOS/OS9000/603/PORTS/MOTRAVEN/ CMDS/BOOTOBJS/DESC/RB1003

Common System Modules List

The Configuration Wizard simplifies the process of building a Coreboot image. Table 2-4 lists the included modules. In this case, the high-level system is to be booted from a hard disk:

These modules are located in the following directory:

```
MWOS/OS9000/PPC/CMDS/BOOTOBJS/ROM
```

Table 2-4 Typical Coreboot Image Contents

Module	Description
`bootsys`	provides booter services.
`console`	provides high-level I/O hooks into low-level console serial driver.
`dbgentry`	provides hooks to low-level debugger server.
`dbgserv`	is a debugger server module.
`excption`	is a service module.
`fdc765`	provides PC style floppy support.
`fdman`	is a target-independent booter support module providing general booting services for RBF file systems.
`flboot`	is a SCSI floptical drive disk booter.
`flshcach`	provides the cache flushing routine.
`fsboot`	is a SCSI TEAC floppy disk drive booter.
`hlproto`	allows user-state debugging.

Table 2-4 Typical Coreboot Image Contents (continued)

Module	Description
`hsboot`	is a SCSI hard disk drive booter.
`ide`	provides target-specific standard IDE support, including PCMCIA ATA PC cards.
`iovcons`	is a hardware independent virtual console driver that provides a telnetd-like interface to the low-level system console.
`llbootp`	is a target-independent BOOTP protocol booter module.
`llip`	is a target-independent internet protocol module.
`llkermit`	is a kermit booter (serial down loader).
`llslip`	is a target-independent serial line internet protocol module. This modules uses the auxiliary communications port driver to perform serial I/O
`lltcp`	is a target-independent transmission control protocol module.
`lludp`	is a target-independent user datagram protocol modules.
`notify`	coordinates use of low-level I/O drivers in system and user-state debugging.
`override`	is a target-independent booter module that enables overriding of the autobooter. If the space bar is pressed within three seconds after booting the target, a boot menu is displayed. Otherwise, booting proceeds with the first autobooter.

Table 2-4 Typical Coreboot Image Contents (continued)

Module	Description
parser	parses key fields from the cnfgdata module and the user parameter fields.
pcman	is a target-independent booter support module providing general booting services for PCF file systems (PC FAT file systems).
protoman	is a target-independent protocol module manager. This module provides the initial communication entry points into the protocol module stack.
restart	restarts boot process.
romboot	locates the OS-9 bootfile in ROM, FLASH, NVRAM.
rombreak	enables break option from the boot menu.
rombug	is a debugger client module.
scsiman	is a target-independent booter support module that provides general SCSI command protocol services
sndp	is a target-independent system-state network debugging protocol module. This module acts as a debugging client on the target, invoking the services of dbgserv to perform debug tasks.
srecord	receives a Motorola S-record format file from the communications port and loads it into memory.
swtimer	is a software timer.

Table 2-4 Typical Coreboot Image Contents (continued)

Module	Description
`type41`	is a primary partition type.
`vcons`	is a console terminal pathlist.