OS-9 for RadiSys ENP-3511 Board Guide

Version 3.2

www.radisys.com

World Headquarters
5445 NE Dawson Creek Drive • Hillsboro, OR
97124 USA
Phone: 503-615-1100 • Fax: 503-615-1121
Toll-Free: 800-950-0044

International Headquarters
Gebouw Flevopoort • Televisieweg 1A
NL-1322 AC • Almere, The Netherlands
Phone: 31 36 5365595 • Fax: 31 36 5365620

RadiSys Microware Communications Software Division, Inc.
1500 N.W. 118th Street
Des Moines, Iowa 50325
515-223-8000

Revision A
November 2001

Copyright and publication information

This manual reflects version 3.2 of Enhanced OS-9 for IXP1200.
Reproduction of this document, in part or whole, by any means, electrical, mechanical, magnetic, optical, chemical, manual, or otherwise is prohibited, without written permission from RadiSys Microware Communications Software Division, Inc.

Disclaimer

The information contained herein is believed to be accurate as of the date of publication. However, RadiSys Corporation will not be liable for any damages including indirect or consequential, from use of the OS-9 operating system, Microware-provided software, or reliance on the accuracy of this documentation. The information contained herein is subject to change without notice.

Reproduction notice

The software described in this document is intended to be used on a single computer system. RadiSys Corporation expressly prohibits any reproduction of the software on tape, disk, or any other medium except for backup purposes. Distribution of this software, in part or whole, to any other party or on any other system may constitute copyright infringements and misappropriation of trade secrets and confidential processes which are the property of RadiSys Corporation and/or other parties. Unauthorized distribution of software may cause damages far in excess of the value of the copies involved.

November 2001

Table of Contents

Chapter 1: Installing and Configuring OS-9

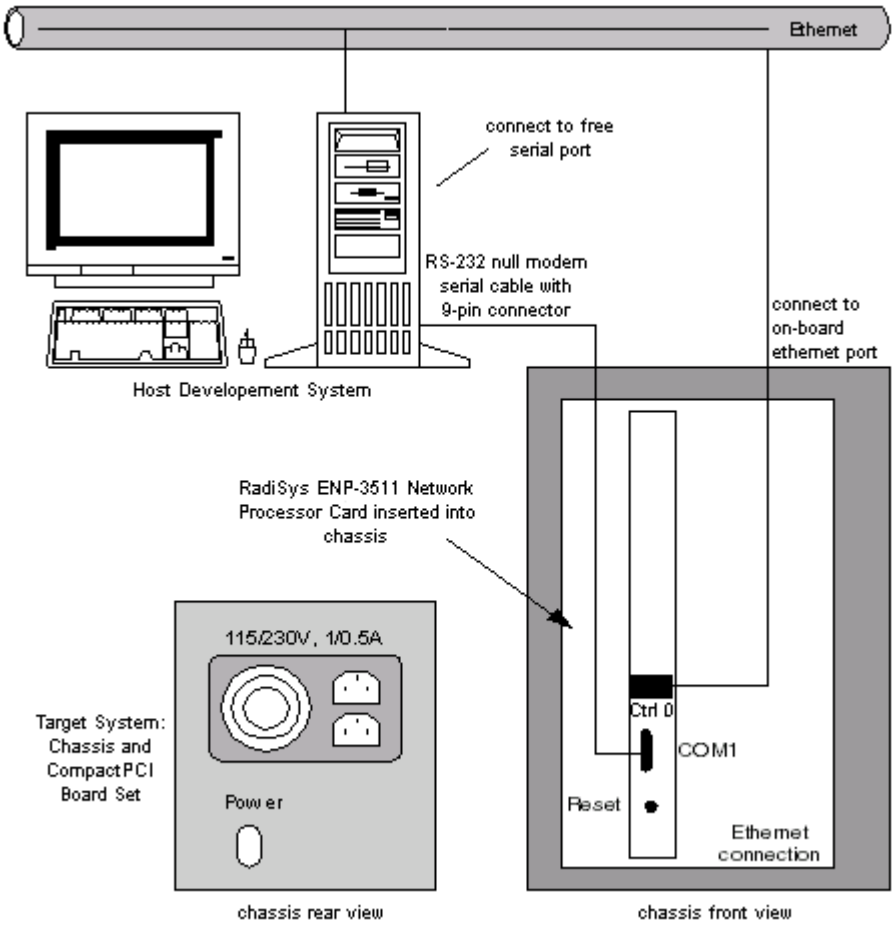
This chapter describes installing and configuring OS-9 on the RadiSys ENP-3511 Embedded Network Processor. It includes the following sections:

- **Development Environment Overview**
- **Requirements and Compatibility**
- **Target Hardware Setup**
- **Connecting the Target to the Host**
- **Building the ROM Image**
- **Burning the Flash Parts**
- **Optional Procedures**

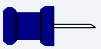
Development Environment Overview

Figure 1-1 shows a typical development environment for the RadiSys ENP-3511. The components shown are the minimum required to develop software with OS-9 and the RadiSys ENP-3511.

Figure 1-1 ENP-3511 Development Environment



Requirements and Compatibility



Note

Before you begin, install ***Enhanced OS-9 for IXP1200*** CD-ROM on your host PC.

Host Hardware Requirements (PC Compatible)

The host PC must have the following minimum hardware characteristics:

- 250MB of free hard disk space
- the recommended amount of RAM for the host operating system
- a CD-ROM drive
- a free serial port
- an Ethernet network card
- access to an Ethernet network

Host Software Requirements (PC Compatible)

The host PC must have the following software installed:

- Enhanced OS-9
- Windows 95, Windows 98, Windows NT 4.0, Windows 2000, or Windows ME
- terminal emulation program

Note

The examples in this document use the terminal emulation program Hyperterminal, which is included with all Windows operating systems.

Target Hardware Requirements

Your reference board requires the following hardware:

- enclosure or chassis with power supply and backplane
- two OS-9 Flash parts
- an RS-232 null modem serial cable with 9-pin connectors
- access to an Ethernet network
- 3511 Rear Transition Module (RTM)

Target Hardware Setup

Installing the Flash Parts

Configuring your reference board consists of installing the Flash parts. To install the parts, complete the following steps:

For More Information

Refer to **Using the Configuration Wizard** section for information on configuring a ROM image that can be burned into the Flash parts.

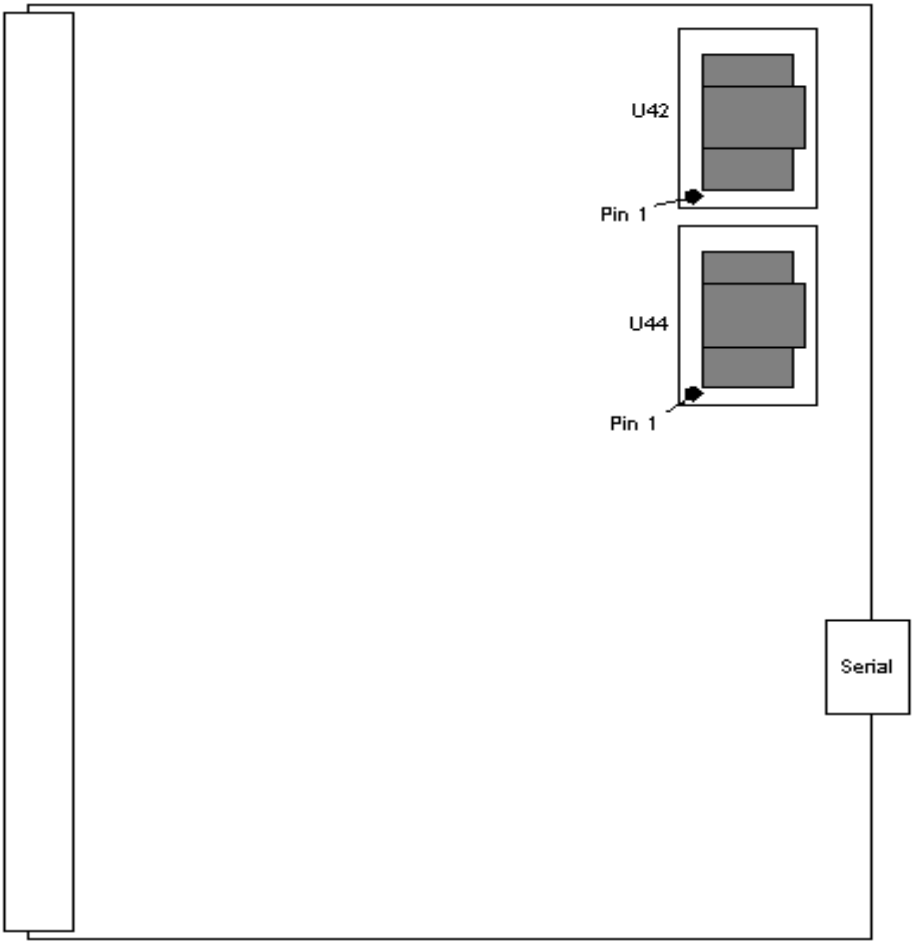
Step 1. With the target system powered down, remove the RadiSys ENP-3511 board from the chassis. Lay the board on a flat, static-free surface.

Step 2. Remove the factory Flash parts from socket U42 and U44. These sockets are located on the back of the board (shown in **Figure 1-2**).

WARNING

Be very careful not to damage the pins on the board or the Flash part. They are easily bent and extremely difficult to repair.

Figure 1-2 RadiSys ENP-3510

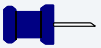


Step 3. Insert the OS-9 formatted Flash parts into the appropriate sockets; U44 is for low 16 bits U42 is for high 16 bits. Be sure the dimple on the Flash part indicating Pin 1 lines up with the asterisk symbol next to the socket on the target board.

Note

Be certain that each flash part is inserted correctly into the appropriate socket. If the parts are inserted improperly, the target will not boot.

Step 4. Once the flash parts have been inserted, close the flash gate by snapping it shut.



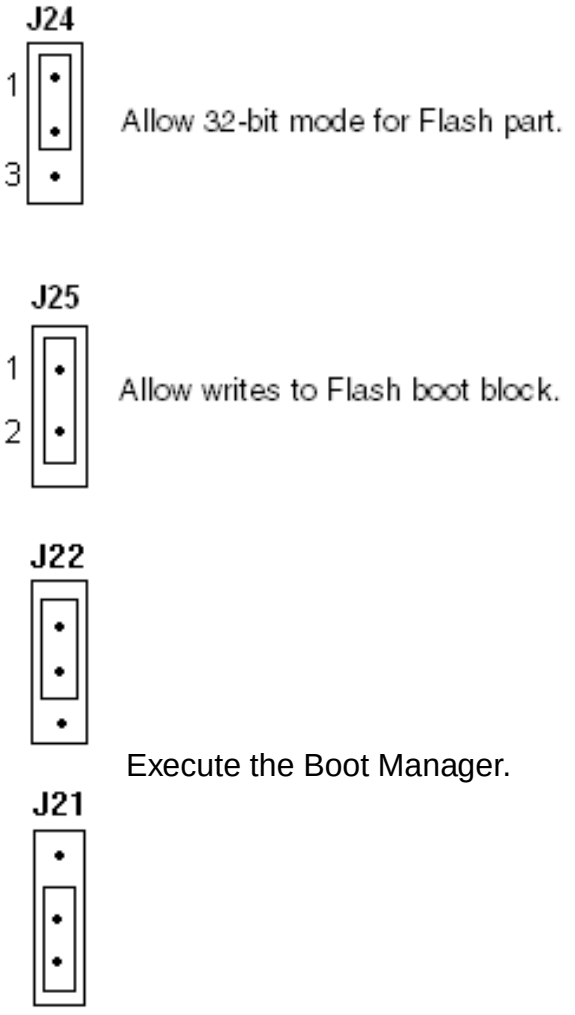
Note

If you need to re-program the Flash devices or create new Flash devices, refer to the **Optional Procedures** section.

Jumper Settings

Verify that the jumper settings on the board match those shown in **Figure 1-3**.

Figure 1-3 Jumper Settings



Connecting the Target to the Host

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

Step 1. Connect the target system to a power supply. Make sure the power switch is in the OFF position.

Step 2. Connect the target system to an Ethernet network. Refer to **Figure 1-1** for a detailed view.

Step 3. Connect the target system to the host system using an RS-232 null modem serial cable with 9-pin connectors. Refer to **Figure 1-1** for a detailed view.

Step 4. On the Windows desktop, click on the `Start` button and select `Programs -> Accessories -> Hyperterminal`.

Step 5. Click the `Hyperterminal` icon.

Step 6. Enter a name for your Hyperterminal session and select an icon for the new session. Click `OK`. A new icon is created with the name of your session associated with it. The settings you choose for this session can be saved for future use.

Step 7. In the **Connect To** dialog, go to the **Connect Using** box and select the communications port with which you will connect to the reference board.

The port you select must be the same port in which you inserted the cable to your host machine.

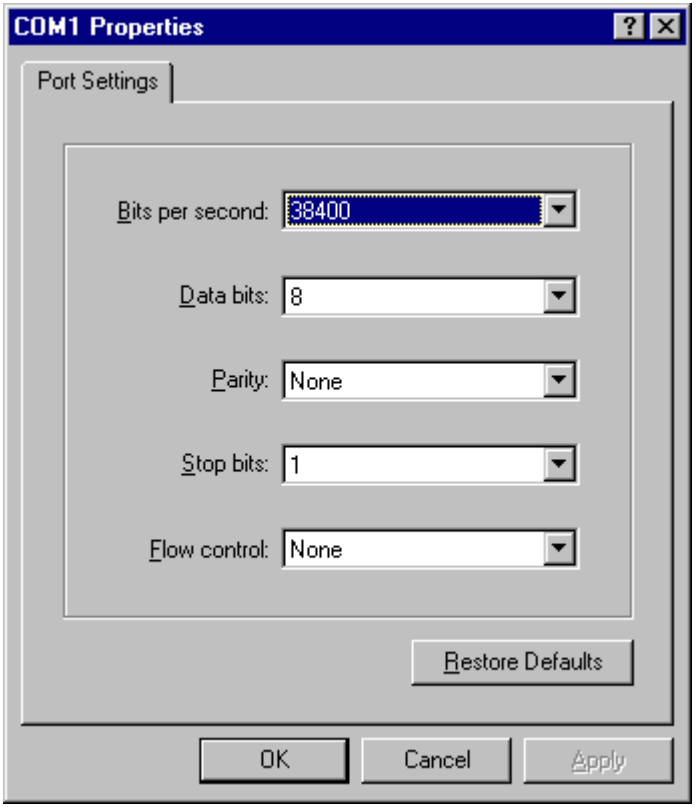
Step 8. Click `OK`.

Step 9. In the **Properties** box, on the **Port Settings** tab, enter the following settings:

```
Bits per second = 38400
Data Bits = 8
Parity = None
Stop bits = 1
Flow control = None
```

Step 10. Click OK.

Figure 1-4 Port Settings



Step 11. Go to the Hyperterminal menu and select `Call -> Connect` from the pull-down menu to establish your terminal session with the reference board. If you are connected, the bottom left of your Hyperterminal screen will display the word *connected*.

Step 12. Turn on the target system. The OS-9 bootstrap message, followed by the OS-9 prompt, $, is displayed in the Hyperterminal window.

At this point, your target system is running a basic OS-9 operating system from the OS-9 Flash part you inserted. Proceed through the following sections to create and download a more sophisticated OS-9 operating system.

Building the ROM Image

The OS-9 ROM Image is a set of files and modules that collectively make up the OS-9 operating system. The specific ROM Image contents can vary from system to system depending on hardware capabilities and user requirements.

To simplify the process of loading and testing OS-9, the ROM Image is generally divided into two parts—the low-level image, called `coreboot`; and the high-level image, called `bootfile`.

Coreboot

The coreboot image is generally responsible for initializing hardware devices and locating the high-level (or bootfile) image as specified by its configuration. Depending on hardware capabilities, the bootfile image could be found on a Flash part, a hard disk, or on an Ethernet network. It is also responsible for building basic structures based on the image it finds and passing control to the kernel to bring up the OS-9 system.

Bootfile

The bootfile image contains the kernel and other high-level modules (initialization module, file managers, drivers, descriptors, and applications). The image is loaded into memory based on the device selected from the boot menu. The bootfile image normally brings up an OS-9 shell prompt, but can be configured to automatically start an application.

Microware provides a Configuration Wizard to create a coreboot image, a bootfile image, or an entire OS-9 ROM Image. The wizard can also be used to modify an existing image. The Configuration Wizard is automatically installed on your host PC during the installation process.

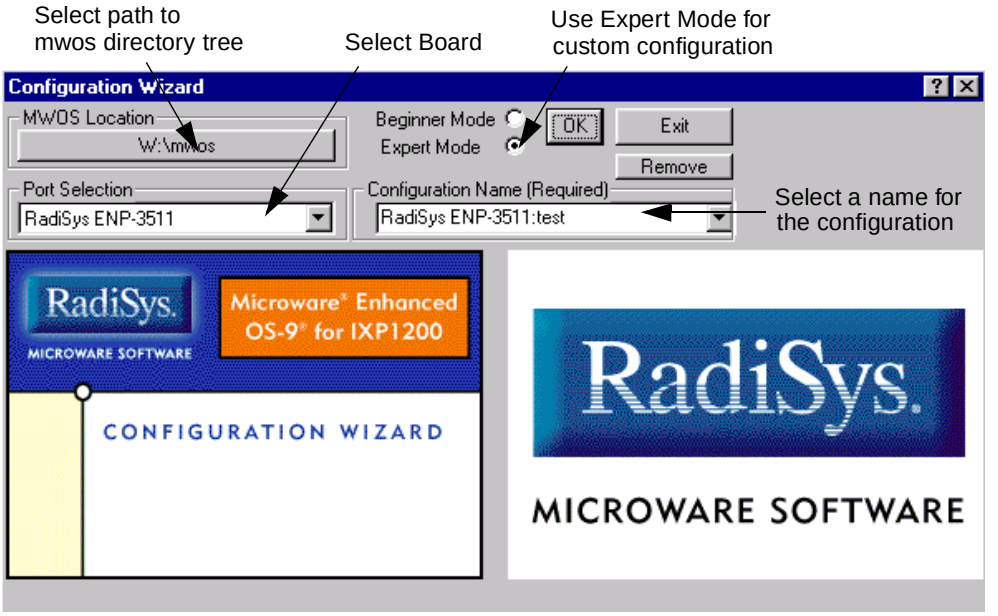
Using the Configuration Wizard

To use the Configuration Wizard, perform the following steps:

Step 1. Click the `Start` button on the Windows desktop.

Step 2. Select `Programs -> Microware -> Enhanced OS-9 for IXP1200 -> Microware Configuration Wizard`. You should see the following opening screen:

Figure 1-5 IXP1200 Configuration Wizard



Step 3. Select the path where the MWOS directory structure can be located by clicking the MWOS location button.

Step 4. Select the target board from the Port Selection pull-down menu.

Step 5. Select a name for your configuration in the Configuration Name field. Your settings will be saved for future use. This enables you to modify the ROM image incrementally, without having to reselect every option for each change.

Step 6. Select `Expert Mode` and click `OK`. The Main Configuration window is displayed. **Expert** mode enables you to make more detailed and specific choices about what modules are included in your ROM image.

Creating the ROM Image

The ROM Image consists of the coreboot image (low-level system files) and the bootfile image (high-level system files). 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—called the ROM Image.

Creating the Coreboot Image

The OS-9 Flash part shipped with Enhanced OS-9 for IXP1200 3.2 include a working coreboot image. No modifications are necessary in the Configuration Wizard.

Creating the Bootfile Image

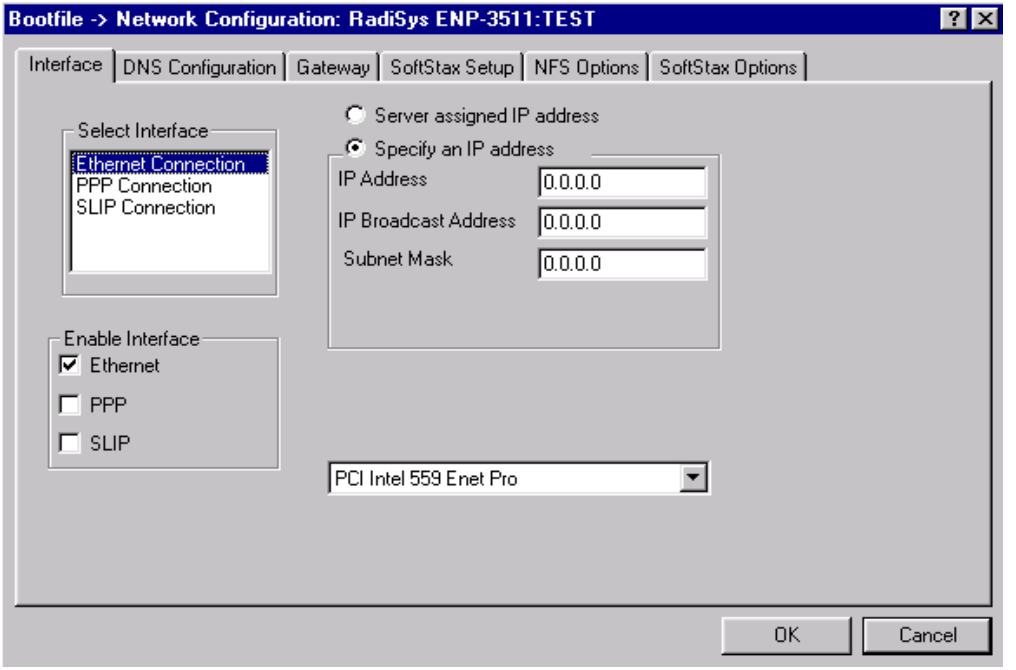
The default settings in the Configuration Wizard have been preset for optimum performance for the RadiSys ENP-3511 Embedded Network Processor. The only modifications required are to enable networking and to change the network settings. The network settings information must be obtained from your network administrator.

Building a ROM Image for the RadiSys ENP-3511

Complete the following steps to build a ROM image for the board:

Step 1. From the Wizard's Main Configuration window, select `Configure -> Bootfile -> Network Configuration`. The **Network Configuration** dialog window should appear with the **Interface** tab displayed (as shown in **Figure 1-6**).

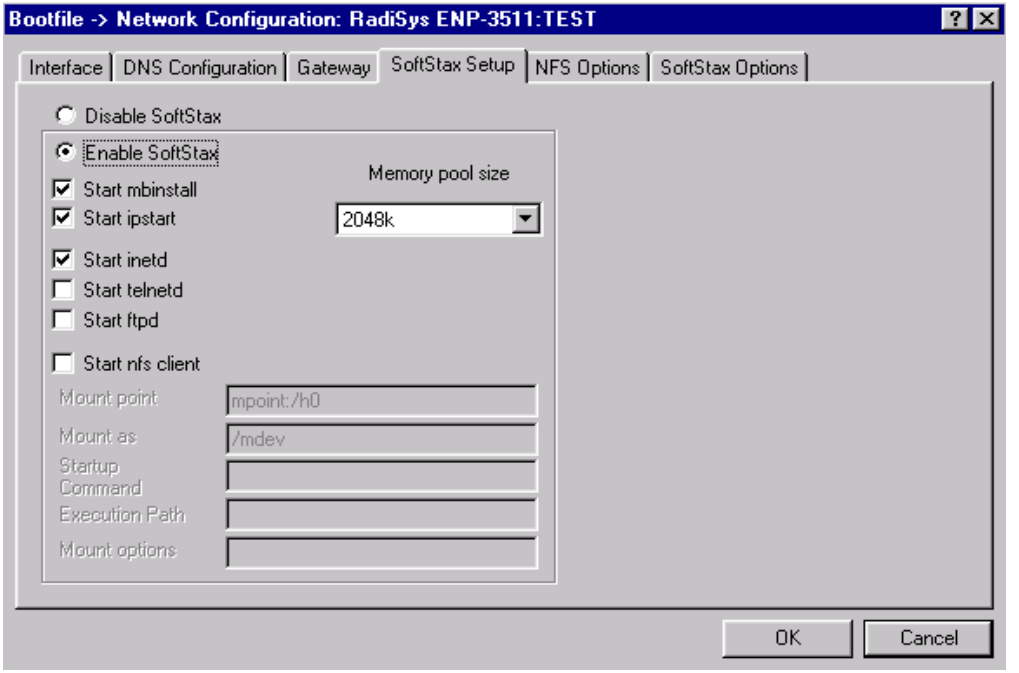
Figure 1-6 Network Configuration—Interface Tab



Step 2. Configure your system as shown in the figure above. The IP address, IP broadcast address, and subnet mask addresses must be obtained from your network administrator.

Step 3. Select the **SoftStax Setup** tab. The **SoftStax Setup** dialog window should appear (as shown in **Figure 1-7**).

Figure 1-7 Network Configuration—SoftStax Setup Tab



Step 4. Configure your system as shown in the above figure.

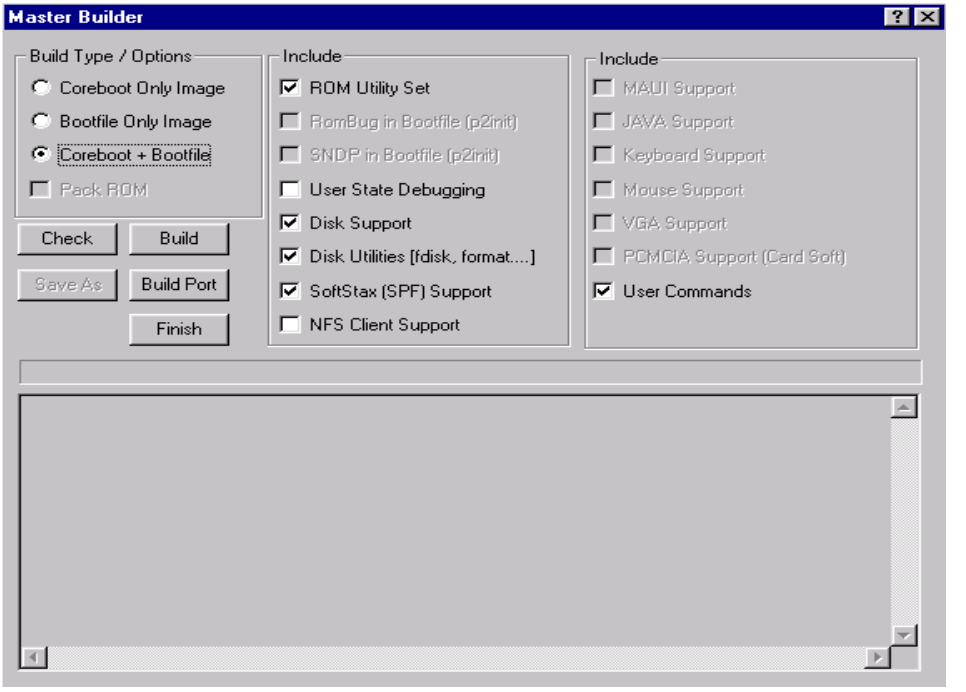
Step 5. Leave the other network configuration options at the default settings. Click OK.

Note

Other network configuration options can be changed in this dialog according to your specific requirements and network.

Step 6. Select `Configure -> Build Image`. The **Master Builder** dialog window appears (as shown in **Figure 1-8**).

Figure 1-8 Master Builder Dialog Window



Step 7. Configure your **Master Builder** options as shown in the figure above.

Step 8. Click `Build`. This builds a ROM image that can be placed on the target. The image, `rom`, is stored in the following default location:

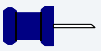
```
\mwos\OS9000\ARMV4\PORTS\ENP3511\BOOTS\INSTALL\PORTBOOT\
```

Clicking `Save As` after the build operation is optional; it enables you to save the ROM image to a location of your choice.

Burning the Flash Parts

You will need to burn the ROM file built in the **Building the ROM Image** section into the Flash parts. This can be done in one of the following three ways:

- OS-9 pflash utility
- RadiSys Boot Monitor
- EPROM burner



Note

To use the EPROM burner to burn the ROM image, you must do one of two things: include the RadiSys Boot Monitor in the image or set up the programmer to ignore the first 2 MB of Flash parts.

Note

The Flash parts are 16-bit path parts. You will need to confirm that your EPROM burner can handle the split properly; it is a 2-way, 16-bit split; thus, the 2-way makes a 32-bit bus. In addition, when burning, the byte swap for each part is selected such that the most significant byte is `1` and the least significant byte is `0`.

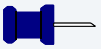
Optional Procedures

The following section provides optional procedures you can perform after installing and configuring OS-9 on your board.

Reprogramming the Flash Parts

The following sections discuss the possible methods for burning the OS-9 image onto your board. The following methods are included:

- **Building a ROM Image with the pflash Utility**
- **Building a ROM Image with the Flash Utility (FUtil)**
- **Building a ROM Image with the EPROM Programmer (using the Boot Monitor)**



Note

If you are placing an OS-9 ROM image on your board for the first time or if your previous OS-9 ROM image did not boot properly, you cannot use the `pflash` utility method.

WARNING

Using the OS-9 `pflash` utility to program a new ROM image into Flash can damage the OS-9 portion of the Flash parts. Any errors made during ROM image modification or during reburning of the Flash part can result in improper booting of the ENP-3511 board. However, the Boot Monitor will still function properly.

Building a ROM Image with the pflash Utility

`pflash` is an OS-9 utility. If OS-9 is currently running on your target, you can use `pflash` to re-program the Flash parts. The following steps detail how to create a new coreboot image and burn the image into Flash using this utility.

Step 1. Click the `Start` button on the Windows desktop.

Step 2. Select `Programs -> Microware -> Enhanced OS-9 IXP1200 -> Microware Configuration Wizard`. The configuration wizard opening screen is displayed (as shown in **Figure 1-5**).

Step 3. Give the image a name in the **Configuration Name** field.

Step 4. Select `Expert Mode` and click `OK`. The Configuration screen is displayed.

Step 5. Make any necessary changes to the coreboot image settings.

(The coreboot options can be found under **Configure** -> **Coreboot** -> **Main Configuration** and the **Configure** -> **Coreboot** -> **Disk Configuration** dialogs. These options enable you to set low-level networking information and select a communication protocol.)

All high-level, bootfile options can be left at the default settings.

Step 6. Select `Configure -> Build Image` to display the **Master Builder** screen.

Step 7. Select the `Coreboot + Bootfile` setting and click `Build`.

Step 8. Click `Save As` to save the coreboot image to a directory of your choosing. If you do not have that directory on the drive, you can create it.

The default location for this file is in the following directory:

```
\mwos\OS9000\ARMV4\PORTS\ENP3511\BOOTS\INSTALL\PORTBOOT\
```

Step 9. Start a DOS shell on the host system.

Step 10. Navigate to the directory in which the OS-9 ROM image, `rom`, is located. The default location for this file is in the following directory:

```
\mwos\OS9000\ARMV4\PORTS\ENP3511\BOOTS\INSTALL\PORTBOOT\
```

Step 11. The image is now created. You should now use FTP and `pflash` to place the new image into the target system's Flash. At the prompt type the following command:

```
ftp <IP address of target>
```

Step 12. At the `ftp>` prompt type the following command:

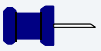
```
bin
```

This designates binary format.

Step 13. At the `ftp>` prompt type the following command:

```
put rom
```

The OS-9 ROM image file is transferred to the target's RAM disk (/r1).



Note

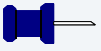
Make sure the RAM disk on the target has been initialized. Otherwise, the file will not show up in the /r1 directory on the target.

Step 14. Open the RadiSys ENP-3511 Hyperterminal window on the Windows host desktop.

Step 15. From the OS-9 prompt ($) type the following command:

```
$ pflash -ri -f=/r1/rom
```

The file is transferred to the target system's Flash memory. `pflash` is configured to copy the `rom` file to the correct address.



Note

If you are only planning to update the bootfile image (os9kboot), you do not have to use the `-ri` option with pflash.

Note

The ENP-3511 board may be shipped with Intel 28I320C2B Flash devices. By default, these Flash devices come up out of reset/power on with the blocks locked. Be sure to include the `-un` option to unlock the blocks.

Building a ROM Image with the Flash Utility (FUtil)

`FUtil` is a DOS utility that can be used with your ENP-3511 target board and a Windows NT host machine. The target and host communicate with each other to use `FUtil`; the host sends commands to the target, which in turn modifies the Flash. This modification is accomplished in the following manner:

- Once it is determined which COM port (serial port) the ENP-3511 target is connected, an attempt is made to connect the host to the target. If the ENP-3511 board has not yet booted, a message similar to the following example will appear:

```
Waiting for connection   0 ...
Waiting for connection   1 ...
Waiting for connection   2 ... Connected
```

- Information displays stating that the Flash has been found on the board. From here, you are prompted to input parameters that dictate how the Flash is to be updated. Once you have entered this information, the new image is sent to the board and the Flash is updated.

Once you understand how the Flash utility works, you can use it to build a ROM image by completing the following steps:

Step 1. Apply power to the board. Press the space bar to stop the auto-boot. You should see the following message:

```
WARNING: IXF440 MAC Address Initialzation failed. sts
=  130
Press space bar to stop auto-boot...
  10


[BootMgr]:
```

Step 2. At the `[BootMgr]:` prompt, type `b0`. You should see the following message:

```
[BootMgr]: b0
FlashUtil, Version 1.0.2
```

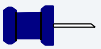
Step 3. At this point the ENP3511 is waiting for communication from the Windows NT host running FUtil. Make sure to disconnect the HyperTerminal session on the ENP3511 before moving to step four.

The FUtil program (FUtil.exe) can be run either before or after the IXP1200 flash utility has been booted.

Step 4. From the Start menu on the Windows desktop, select `Programs -> Microware -> Enhanced OS-9 IXP1200 -> Microware Configuration Wizard`. The Configuration Wizard opening screen is displayed (as shown in Figure 1-5).

Step 5. Name your configuration, select `Expert Mode`, and click `OK`. The Configuration Wizard main window is displayed.

Step 6. Make any necessary changes to the coreboot and/or bootfile image settings.



Note

All high-level, bootfile options can be left at the default settings unless you need or want to change the high-level network addresses.

Step 7. Select `Configure -> Build Image` to display the **Master Builder** dialog.

Step 8. De-select the **Compress Bootfile** check box. (The current version of FUtil does not have support for this.)

Step 9. Select the `Coreboot + Bootfile` setting and click `Build`.

Step 10. Click `Save As` to save the ROM image to a directory of your choice.

The default location for this file is in the following directory:

```
\MWOS\OS9000\ARMV4\PORTS\ENP3511\BOOTS\INSTALL\PORTBOOT\
```

Step 11. Start a DOS shell on the host system and navigate to the directory in which the OS-9 ROM image, `ROM`,is located.

The default location for this file is in the following directory:

```
\MWOS\OS9000\ARMV4\PORTS\ENP3511\BOOTS\INSTALL\PORTBOOT\
```

Step 12. Type the following command:

```
futil
```

If FUtil has correctly updated the Flash, a message similar to the following will display:

```
Flash Utility, Version 1.1
Numeric parameters (other than the bank and COM port) should be entered
as hexadecimal numbers without the leading '0x'.
Enter a blank line to accept the default value.
Enter a period ('.') to abort the program.
Use Serial Port (Yes):
Comm port (1):
FlashUtil, Version 1.0.2
CPU Revision: 6901C123
Flash Width: 32
Flash Bank Size: 4096 KBytes
```

```
Flash Type Bank[0]: 28F320C3-B
Flash Type Bank[1]: 28F320C3-B
Flash Type Bank[2]: 28F320C3-B
Flash Type Bank[3]: 28F320C3-B
Starting at bank 0
Filename (): rom
Starting offset in flash (000000): 200000
Ending offset in flash (1C0000): 800000
Starting offset in file (000000):
Do you want to re-program the FPGA Data partition (Yes): Yes
Sending data to remote system...
 Percent complete: 100
Updating flash...
 Percent Complete: 100
done
```

Step 13. At this point you can reconnect to your ENP3511 using a Hyperterminal session and reboot the ENP3511 board.

Building a ROM Image with the EPROM Programmer (using the Boot Monitor)

The following steps detail how to create a ROM image using the EPROM Programmer and the RadiSys Boot Monitor.

Step 1. Click the `Start` button on the Windows desktop.

Step 2. Select `Programs -> Microware -> Enhanced OS-9 IXP1200 -> Microware Configuration Wizard`. The Configuration Wizard opening screen is displayed (as shown in **Figure 1-5**).

Step 3. Give the image a name in the **Configuration Name** field.

Step 4. Select `Expert Mode` and click `OK`. The configuration screen is displayed.

Step 5. Make any necessary changes to the coreboot and/or bootfile image settings.

Step 6. Select `Configure -> Build Image` to display the **Master Builder** screen.

Step 7. Select the `Coreboot + Bootfile` setting and click `Build`.

Step 8. Click `Save As` to save the coreboot image to a directory of your choosing. If you do not have that directory on the drive, you can create it.

Step 9. Transfer the ROM image to the Flash device with the EPROM programmer.

For More Information

When you are done creating the ROM image, refer to your EPROM programmer's instructions to learn how to load the image into the Flash device.

The Boot Manager

The Boot Manager (BootMgr) allows you to select which of the operating systems residing in Flash will be booted by the board. The BootMgr is accesible via a terminal connected to the board's serial port. Your current settings are saved into the Flash; thus, if no interaction is detected by the BootMgr, the current default operating system is automatically booted.

Note

The Boot Manager region in Flash is write protected and the reflashing tools described here will not update that region of the flash memory.

When the BootMgr starts, it prints out a banner on the serial port (at 38400 Baud):

```
Press space bar to stop auto-boot...
```

```
     10
```

This value (which is configurable) then counts down to zero. If it reaches zero before you press the space bar, the current default operating system will be booted. After auto-booting is stopped, the BootMgr prompt is displayed. You can then enter a command. To display all possible commands, type h, as shown below:

```
[BootMgr]: h
BootMgr commands:
p : Print boot parameters
c : Change boot parameters
b : Boot with current parameters
b <os> : Boot given os without changing parameters
h : Print this help message
```

Boot Manager Commands

The following section describes some of the available Boot Manager commands:

The print command p will display the current parameters:

```
[BootMgr]: p
BootMgr Version 1.0.2
CPU Revision 6901C123
OS list:
     0 Flash Utility
     1 Monitor
     2 Reserved
     3 VxWorks
     4 OS-9
Default OS: 4
Countdown value: 10
Disable initial display: 0

[BootMgr]:
```

The four parameters that may be varied include:

default operating system	This is the OS that will be booted if the autoboot countdown gets to zero. It's value is limited to the displayed list.
countdown value:	This is the number of seconds that the BootMgr will wait before auto-booting the default OS. The tradeoff is that with smaller values, there will be less delay in booting the desired OS, but the user will also have to be quicker in changing the boot parameters. This parameter is limited to values between 1 and 60 inclusive; i.e. a value of 0 is not allowed.
disable initial display:	If this value is set to 1, then the countdown display will not be printed. The BootMgr will still count down before auto booting, but no messages will be displayed unless the autoboot is stopped.

These parameters can be modified with the change command c. This command will print the current parameters and then prompt the user for new values. A blank line will leave the current value unchanged. This dialog looks similar to:

```
BootMgr Version 1.0.2
CPU Revision 6901C123
OS list:
     0 Flash Utility
     1 Monitor
     2 Reserved
     3 VxWorks
     4 OS-9
Disable initial display: 0
Enter blank line to leave value unchanged
Default OS:
Countdown value: 3
```

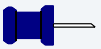
```
Disable initial display:


[BootMgr]:
```

In the example above, the countdown value was changed from 10 to 3, and the other parameters reamained unchanged.

The boot command b will boot the default operating system. Alternately, it can be given a numeric argument (such as `b 0`), which will boot the indicated operating system without changing the default. This could be used, for example, to boot the flash utility without modifying the default operating system.

Creating a High-Level/Low-Level ROM Image

The following procedures describe how to configure a ROM image that supports both a low-level (for Hawk system state debugging) and high-level (for Hawk user state debugging and Intel® Developer Workbench usage) configuration. Complete the following steps to create this image:



Note

These procedures assume that you have already completed the procedures described in the **Building the ROM Image** and **Burning the Flash Parts** sections.

Step 1. In the **Main Configuration** window of the Wizard, select `Configure -> Coreboot -> Main Configuration`. Select the **Ethernet** tab.

Enable the **Add to Boot Menu** check box.

Step 2. Select `Configure -> Bootfile -> Disk Configuration -> Init Options` and then select the following items:

- **Mshell** radio button
- **/dd** radio button
- **User** radio button

  From the **User** button, remove the following string from the **Parameter** list:

  ```
  mbinstall ;ipstart; inetd <>>>/nil&;
  ```

Step 3. Select `Sources -> Port -> User`. Add the following lines:

```
* hlproto provides user level code access to protoman
../../../../../../ARMV4/CMDS/BOOTOBJS/ROM/hlproto
*
* low-level user-state debugging modules
../../../../../../ARMV4/CMDS/undpd
../../../../../../ARMV4/CMDS/undpdc
*
*   sndp - Hawk system-state debug client module
../../../../../../ARMV4/CMDS/BOOTOBJS/ROM/sndp
```

Step 4. Select `Configure -> Build`. The **Master Builder** dialog window appears.

Step 5. Enable the **User State Debugging Modules** box and select `Coreboot + Bootfile`.

Step 6. Click `Build`.

Step 7. Start a DOS shell on the host system.

Step 8. Navigate to the directory in which the OS-9 ROM image, `rom`, is located. The default location for this file is in the following directory:

```
\mwos\OS9000\ARMV4\PORTS\ENP3511\BOOTS\INSTALL\PORTBOOT\
```

Step 9. At the prompt type the following command:

```
ftp <IP address of target>
```

Step 10. At the `ftp>` prompt type the following command:

```
bin
```

This designates binary format.

Step 11. At the `ftp>` prompt type the following command:

```
put rom
```

The OS-9 ROM image is transferred to the target's RAM disk (/r1).

Step 12. Open the RadiSys ENP-3511 Hyperterminal window on the Windows host desktop.

Step 13. From the OS-9 prompt ($) type the following command:

```
$ pflash -ri -f=/r1/rom
```

The OS-9 ROM image is transferred to the target system's Flash memory. `pflash` is configured to copy the file to the correct address.

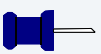
The target system now contains all necessary modules for both a high-level and low-level configuration. However, these configurations cannot be used simultaneously. As the target system is boots, you must choose between a high-level or low-level configuration, according to your needs. The following commands must be entered at the OS-9 prompt after booting for each configuration:

- Low-Level Configuration Procedures

```
$ p2init hlproto
$ p2init sndp
$ undpd <>>>/nil&
```

- High-Level Configuration Procedures

```
$ mbinstall
$ ipstart
$ inetd<>>>/nil&
$ spfndpd <>>>/nil&
```



Note

The low- and high-level ethernet drivers cannot be initialized at the same time. If you want simultaneous support for both system state debugging and user state debugging, choose from the following options:

- Use low-level SLIP for system state debugging.
- Use ROMBug for system state debugging.
- Configure high-level debugging without high-level ethernet support.

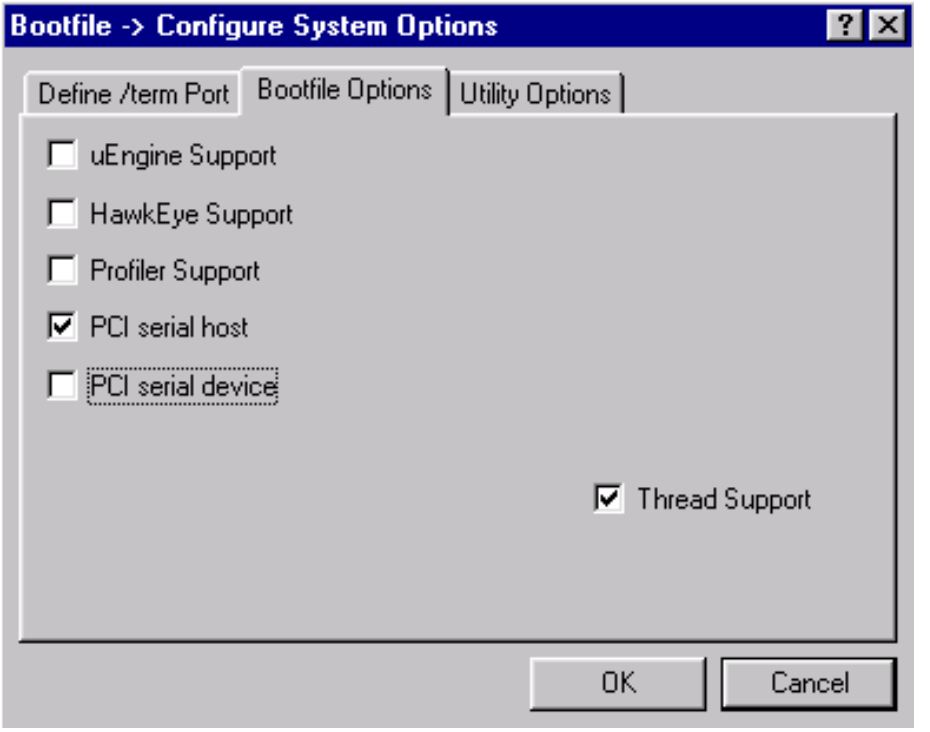
PCI Backplane Communications

To communicate with the PCI backplane, complete the following steps:

Step 1. From the Configuration Wizard, select `Configure` -> `Bootfile` -> `Configure System Options`. Select the **Bootfile Options** tab.

Step 2. Select either the **PCI serial host** or **PCI serial device** check box (as shown in **Figure 1-9**).

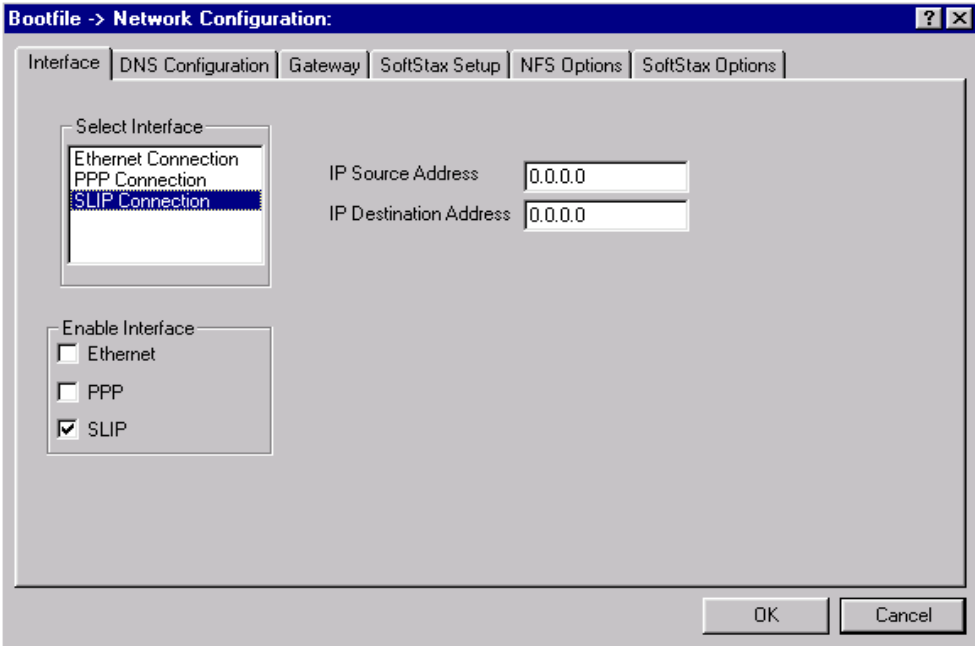
Figure 1-9 Bootfile Options tab



Step 3. Select `Configure` -> `Bootfile` -> `Network Configuration`. Select the **Interface** tab.

Step 4. Set the Interface tab to configure a slip interface (as shown in **Figure 1-10**).

Figure 1-10 Interface tab



Because the only IXP host supported is the Intel Ethernet Eval board, you will need to port the driver to whichever architecture you will be using as your host.

Chapter 2: Board Specific Reference

This chapter contains porting information that is specific to the RadiSys ENP-3511 Embedded Network Processor. It includes the following sections:

- **Boot Options**
- **The Fastboot Enhancement**
- **OS-9 Vector Mappings**
- **Port Specific Utilities**
- **Intel Work Bench Daemons**

For More Information

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

Boot Options

The default boot options for the RadiSys ENP-3511 are listed below. The boot options can be selected by hitting the space bar during system bootup when the following message appears on the serial console:

```
Now trying to Override autobooters
```

The configuration of these booters can be changed by altering the `default.des` file, which is located in the following directory:

```
mwos\OS9000\ARMV4\PORTS\IXP1200\ROM
```

Booters can be configured to be either menu or auto booters. The auto booters automatically attempt to boot in the same order as listed in the auto booter array. Menu booters from the defined menu booter array are chosen interactively from the console command line after the boot menu is displayed.

Booting from Flash

When the `rom_cnfg.h` file has a ROM search list defined, the options `ro` and `lr` appear in the boot menu. If no search list is defined `N/A` appears in the boot menu. If an OS-9 bootfile is programmed into Flash memory in the address range defined in the port's `default.des` file, the system can boot and run from Flash.

`rom_cnfg.h` is located in the following directory:

```
mwos\os9000\ARMV4\PORTS\IXP1200\ROM\ROMCORE
```

`ro`	ROM boot—the system runs from the Flash bank.
`lr`	load to RAM—the system copies the Flash image into RAM and runs from there.

Booting from on-Board Ethernet Interface

The system can boot using the BootP protocol with an Ethernet IntelPro 100 and `eb` option.

`eb`	Ethernet boot: A PCI card that supports Ethernet will use the bootp protocol to transfer a bootfile into RAM and the systems runs from there.

Booting over a Serial Port via kermit

The system can download a bootfile in binary form over its serial port at speeds up to 115200 using the kermit protocol. The speed of this transfer depends of the size of the bootfile, but it usually takes at least three minutes to complete. Dots on the console will show the progress of the boot.

`ker`	kermit boot: The `os9kboot` file is sent via the kermit protocol into system RAM and runs from there.

Restart Booter

The restart booter enables a way to restart the bootstrap sequence.

`q`	quit: Quit and attempt to restart the booting process.

Break Booter

The break booter allows entry to the system level debugger (if one exists). If the debugger is not in the system the system will reset.

`break`	break: Break and enter the system level debugger Rombug.

Sample Boot Session and Messages

Below is an example boot of the RadiSys ENP-3511 using the `lr` boot option.

```
OS-9 Bootstrap for the ARM (Edition 64)

Now trying to Override autobooters.

Press the spacebar for a booter menu


BOOTING PROCEDURES AVAILABLE ---------- <INPUT>

Boot embedded OS-9 in-place -------- <bo>
Copy embedded OS-9 to RAM and boot - <lr>
Load bootfile via kermit Download ----- <ker>
Enter system debugger ----------------- <break>
Restart the System -------------------- <q>

Select a boot method from the above menu: lr

Now searching memory ($00040000 - $001fffff) for an OS-9
Kernel...

An OS-9 kernel was found at $00040000
A valid OS-9 bootfile was found.
$
$ pciv

BUS:DV:FU  VID  DID  CMD  STAT CLASS  RV CS IL IP
-----------------------------------------------
000:00:00  8086 1200 0017 2200 0b4001 00 00 6e 0e IXP1200
Processor [S]
000:05:00  11ad 0002 0007 0280 020000 20 00 6e 01 Network
Controller [S]

$ ipstart
$ ping 172.16.3.202
PING 172.16.3.202 (172.16.3.202): 56 data bytes
64 bytes from 172.16.3.202: ttl=255 time=10 ms
```

The Fastboot Enhancement

The Fastboot enhancements to OS-9 provide faster system bootstrap performance to embedded systems. The normal bootstrap performance of OS-9 is attributable to its flexibility. OS-9 handles many different runtime configurations to which it dynamically adjusts during the bootstrap process.

The Fastboot concept consists of informing OS-9 that the defined configuration is static and valid. These assumptions eliminate the dynamic searching OS-9 normally performs during the bootstrap process and enables the system to perform a minimal amount of runtime configuration. As a result, a significant increase in bootstrap speed is achieved.

Overview

The Fastboot enhancement consists of a set of flags that control the bootstrap process. Each flag informs some portion of the bootstrap code that a particular assumption can be made and that the associated bootstrap functionality should be omitted.

The Fastboot enhancement enables control flags to be statically defined when the embedded system is initially configured as well as dynamically altered during the bootstrap process itself. For example, the bootstrap code could be configured to query dip switch settings, respond to device interrupts, or respond to the presence of specific resources which would indicate different bootstrap requirements.

In addition, the Fastboot enhancement's versatility allows for special considerations under certain circumstances. This versatility is useful in a system where all resources are known, static, and functional, but additional validation is required during bootstrap for a particular instance, such as a resource failure. The low-level bootstrap code may respond to some form of user input that would inform it that additional checking and system verification is desired.

Implementation Overview

The Fastboot configuration flags have been implemented as a set of bit fields. An entire 32-bit field has been dedicated for bootstrap configuration. This four-byte field is contained within the set of data structures shared by the ModRom sub-components and the kernel. Hence, the field is available for modification and inspection by the entire set of system modules (high-level and low-level). Currently, there are six bit flags defined with eight bits reserved for user-definable bootstrap functionality. The reserved user-definable bits are the high-order eight bits (31-24). This leaves bits available for future enhancements. The currently defined bits and their associated bootstrap functionality are listed below:

B_QUICKVAL

The `B_QUICKVAL` bit indicates that only the module headers of modules in ROM are to be validated during the memory module search phase. This causes the CRC check on modules to be omitted. This option is a potential time saver, due to the complexity and expense of CRC generation. If a system has many modules in ROM, where access time is typically longer than RAM, omitting the CRC check on the modules will drastically decrease the bootstrap time. It is rare that corruption of data will ever occur in ROM. Therefore, omitting CRC checking is usually a safe option.

B_OKRAM

The `B_OKRAM` bit informs both the low-level and high-level systems that they should accept their respective RAM definitions without verification. Normally, the system probes memory during bootstrap based on the defined RAM parameters. This allows system designers to specify a possible RAM range, which the system validates upon startup. Thus, the system can accommodate varying amounts of RAM. In an embedded system where the RAM limits are usually statically defined and presumed to be functional, however, there is no need to validate the defined RAM list. Bootstrap time is saved by assuming that the RAM definition is accurate.

B_OKROM

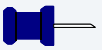
The B_OKROM bit causes acceptance of the ROM definition without probing for ROM. This configuration option behaves like the B_OKRAM option, except that it applies to the acceptance of the ROM definition.

B_1STINIT

The B_1STINIT bit causes acceptance of the first init module found during cold-start. By default, the kernel searches the entire ROM list passed up by the ModRom for init modules before it accepts and uses the init module with the highest revision number. In a statically defined system, time is saved by using this option to omit the extended init module search.

B_NOIRQMASK

The B_NOIRQMASK bit informs the entire bootstrap system that it should not mask interrupts for the duration of the bootstrap process. Normally, the ModRom code and the kernel cold-start mask interrupts for the duration of the system startup. However, some systems that have a well defined interrupt system (i.e. completely calmed by the sysinit hardware initialization code) and also have a requirement to respond to an installed interrupt handler during system startup can enable this option to prevent the ModRom and the kernel cold-start from disabling interrupts. This is particularly useful in power-sensitive systems that need to respond to “power-failure” oriented interrupts.



Note

Some portions of the system may still mask interrupts for short periods during the execution of critical sections.

B_NOPARITY

If the RAM probing operation has not been omitted, the B_NOPARITY bit causes the system to not perform parity initialization of the RAM. Parity initialization occurs during the RAM probe phase. The B_NOPARITY option is useful for systems that either require no parity initialization at all or systems that only require it for “power-on” reset conditions. Systems that only require parity initialization for initial “power-on” reset conditions can dynamically use this option to prevent parity initialization for subsequent “non-power-on” reset conditions.

Implementation Details

This section describes the compile-time and runtime methods by which the bootstrap speed of the system can be controlled.

Compile-time Configuration

The compile-time configuration of the bootstrap is provided by a pre-defined macro (BOOT_CONFIG), which is used to set the initial bit-field values of the bootstrap flags. You can redefine the macro for recompilation to create a new bootstrap configuration. The new over-riding value of the macro should be established by redefining the macro in the rom_config.h header file or as a macro definition parameter in the compilation command.

The rom_config.h header file is one of the main files used to configure the ModRom system. It contains many of the specific configuration details of the low-level system. Below is an example of how you can redefine the bootstrap configuration of the system using the BOOT_CONFIG macro in the rom_config.h header file:

```
#define BOOT_CONFIG (B_OKRAM + B_OKROM + B_QUICKVAL)
```

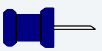
Below is an alternate example showing the default definition as a compile switch in the compilation command in the makefile:

```
SPEC_COPTS = -dNEWINFO –dNOPARITYINIT –dBOOT_CONFIG=0x7
```

This redefinition of the `BOOT_CONFIG` macro results in a bootstrap method that accepts the RAM and ROM definitions without verification, and also validates modules solely on the correctness of their module headers.

Runtime Configuration

The default bootstrap configuration can be overridden at runtime by changing the `rinf->os->boot_config` variable from either a low-level P2 module or from the `sysinit2()` function of the `sysinit.c` file. The runtime code can query jumper or other hardware settings to determine what user-defined bootstrap procedure should be used. An example P2 module is shown below.



Note

If the override is performed in the `sysinit2()` function, the effect is not realized until after the low-level system memory searches have been performed. This means that any runtime override of the default settings pertaining to the memory search must be done from the code in the P2 module code.

```
#define NEWINFO
#include <rom.h>
#include <types.h>
#include <const.h>
#include <errno.h>
#include <romerrno.h>
#include <p2lib.h>

error_code p2start(Rominfo rinf, u_char *glbls)
{
    /* if switch or jumper setting is set… */
    if (switch_or_jumper == SET) {
        /* force checking of ROM and RAM lists */
        rinf->os->boot_config &= ~(B_OKROM+B_OKRAM);
    }
    return SUCCESS;
}
```

OS-9 Vector Mappings

This section contains the OS-9 vector mappings for the RadiSys ENP-3511 Embedded Network Processor.

The ARM standard defines exceptions 0x0-0x7. The OS-9 system maps these one-to-one. External interrupts from vector 0x6 are expanded to the virtual vector range shown below by the `irqixp1200` module.

For More Information

Refer to the ***RadiSys ENP-3511 Network Processor Programer's Reference*** for further information on individual sources.

Table 2-1 OS-9 IRQ Assignment for the RadiSys ENP-3511

OS-9 IRQ #	ARM Function
0x0	Processor Reset
0x1	Undefined Instruction
0x2	Software Interrupt
0x3	Abort on Instruction Prefetch
0x4	Abort on Data Access
0x5	Unassigned/Reserved

Table 2-1 OS-9 IRQ Assignment for the RadiSys ENP-3511

OS-9 IRQ #	ARM Function
0x6	External Interrupt
0x7	Fast Interrupt
0x8	Alignment error Form of Data abort

Table 2-2 RadiSys ENP-3511 Specific Functions

OS-9 IRQ #	ENP-3511 Specific Function
0x40	MicroEngine 0, thread 0 (FBI block sources 0-28)
0x41	MicroEngine 0, thread 1
0x42	MicroEngine 0, thread 2
0x43	MicroEngine 0, thread 3
0x44	MicroEngine 1, thread 0
0x45	MicroEngine 1, thread 1
0x46	MicroEngine 1, thread 2
0x47	MicroEngine 1, thread 3
0x48	MicroEngine 2, thread 0
0x49	MicroEngine 2, thread 1
0x4a	MicroEngine 2, thread 2

Table 2-2 RadiSys ENP-3511 Specific Functions (continued)

OS-9 IRQ #	ENP-3511 Specific Function
0x4b	MicroEngine 2, thread 3
0x4c	MicroEngine 3, thread 0
0x4d	MicroEngine 3, thread 1
0x4e	MicroEngine 3, thread 2
0x4f	MicroEngine 3, thread 3
0x50	MicroEngine 4, thread 0
0x51	MicroEngine 4, thread 1
0x52	MicroEngine 4, thread 2
0x53	MicroEngine 4, thread 3
0x54	MicroEngine 5, thread 0
0x55	MicroEngine 5, thread 1
0x56	MicroEngine 5, thread 2
0x57	MicroEngine 5, thread 3
0x58	Debug interrupt 0
0x59	Debug interrupt 1
0x5a	Debug interrupt 2
0x5b	CINT pin

Table 2-2 RadiSys ENP-3511 Specific Functions (continued)

OS-9 IRQ #	ENP-3511 Specific Function
0x5c	Reserved (PCI block sources 0-32)
0x5d	Soft Interrupt
0x5e	Reserved
0x5f	Reserved
0x60	Timer 1
0x61	Timer 2
0x62	Timer 3
0x63	Timer 4
0x64	Reserved
0x65	Reserved
0x66	Reserved
0x67	Reserved
0x68	Reserved
0x69	Reserved
0x6a	Reserved
0x6b	Door Bell from host
0x6c	DMA channel 1

Table 2-2 RadiSys ENP-3511 Specific Functions (continued)

OS-9 IRQ #	ENP-3511 Specific Function
0x6d	DMA channel 2
0x6e	PCI_IRQ_1 (External PCI interrupts)
0x6f	Reserved
0x70	DMA1 not busy
0x71	DMA2 not busy
0x72	Start BIST
0x73	Received SERR
0x74	Reserved
0x75	I20 inbound post_list
0x76	Power management
0x77	Discard timer expired
0x78	Data parity error detected
0x79	Received master abort
0x7a	Received target abort
0x7b	Detected PCI Parity error
0x7c	SRAM interrupt (SRAM block source)

Table 2-2 RadiSys ENP-3511 Specific Functions (continued)

OS-9 IRQ #	ENP-3511 Specific Function
0x7d	RTC (RTC block source)
0x7e	SDRAM (SDRAM block source)
0x7f	UART (UART block source)

Fast Interrupt Vector (0x7)

The ARM4 defined fast interrupt (FIQ) mapped to vector 0x7 is handled differently by the OS-9 interrupt code and can not be used as freely as the external interrupt mapped to vector 0x6. To make fast interrupts as quick as possible for extremely time critical code, no context information is saved on exception (except auto hardware banking) and FIQs are never masked. This requires any exception handler to save and restore its necessary context if the FIQ mechanism is to be used. This requirement means that a FIQ handler's entry and exit points must be in assembly, as the C compiler will make assumptions about context. In addition, no system calls are possible unless a full C ABI context save has been performed first. The OS-9 IRQ code for the SA1100 has assigned all interrupts as normal external interrupts. It is up to the user to re-define a source as an FIQ to make use of this feature.

Port Specific Utilities

Utilities for the RadiSys ENP-3511 are located in the following directory:

`MWOS/OS9000/ARMV4/PORTS/IXP1200/CMDS`

The following port specific utilities are included:

`dmppci`	peeks PCI device information
`pciv`	displays board PCI bus information
`pflash`	programs onboard Flash
`setpci`	pokes PCI device settings

dmppci — Show PCI Information

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 5 0 0x40

    PCI DUMP Bus:0 Dev:5 Func:0 Size:64
    -----------------------------------

VID  DID  CMD  STAT CLASS  RV CS IL IP LT HT BI MG ML SVID SDID
---  ---- ---- ---- -----  -- -- -- -- -- -- -- -- -- ---- ----
11ad 0002 0007 0280 020000 20 00 6e 01 00 00 00 00 00 1385 f004

BASE[0]  BASE[1]  BASE[2]  BASE[3]  BASE[4]  BASE[5]  CIS_P    EXROM
-------- -------- -------- -------- -------- -------- -------- --------
54000001 7fffff00 00000000 00000000 00000000 00000000 00000000 00000000

Offset 00 01 02 03 04 05 06 07 08 09 0a 0b 0c 0d 0e 0f
------------------------------------------------------
0000   ad 11 02 00 07 00 80 02 20 00 00 02 00 00 00 00
0010   01 00 00 54 00 ff ff 7f 00 00 00 00 00 00 00 00
0020   00 00 00 00 00 00 00 00 00 00 00 00 85 13 04 f0
0030   00 00 00 00 00 00 00 00 00 00 00 00 6e 01 00 00
```

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.

EXAMPLES

When using the `pciv` command with a RadiSys ENP-3511, the following information is displayed:

```
$ pciv -a

BUS:DV:FU  VID  DID  CMD  STAT CLASS  RV CS IL IP
-------------------------------------------------
000:00:00  8086 1200 0017 2200 0b4001 00 00 6e 0e
(NC) [32-bit] base_addr[0] = 0x40000008  PCI/MEM 0x60000008 Size = 0x00100000
(C)  [32-bit] base_addr[1] = 0x50000001  PCI/IO  0x54000000 Size = 0x00000080
(NC) [32-bit] base_addr[2] = 0xc0000008  PCI/MEM 0xe0000008 Size = 0x02000000
IXP1200 Processor [S]

BUS:DV:FU  VID  DID  CMD  STAT CLASS  RV CS IL IP
-------------------------------------------------
000:05:00  11ad 0002 0007 0280 020000 20 00 6e 01
(C)  [32-bit] base_addr[0] = 0x54000001  PCI/IO  0x54000000 Size = 0x00000100
(C)  [32-bit] base_addr[1] = 0x7fffff00  PCI/MEM 0x7fffff00 Size = 0x00000100
Network Controller [S]
```

The `pciv` command in the previous example reports configuration information related to specific hardware attached to the system.

```
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 displayed along with the size of the device blocks in use.

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 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.

pflash — Program Strata Flash

Syntax

```
pflash [options]
```

Options

`-f[=]filename`	input filename
`-eu`	erase used space only (default)
`-ew`	erase whole Flash
`-ne`	do not erase Flash
`-ri`	Rom image--allows overwrite of boot block
`-b[=]addr`	specify base address of Flash (hex) for part identification (replaces -r,-p0,-p1)
`-s[=]addr`	specify write/erase address of Flash (hex) defaults to base address)
`-i`	print out information on Flash
`-nv`	do not verify erase or write
`-q`	no progress indicator
`-un`	unlock Flash parts
`-l`	limit bank to 2MB
`-rl`	print lock status, use with `-i`

Description

The pflash utility allows the programming of Intel Strata Flash parts. The primary use will be in the burning of the OS-9 ROM image into the on-board Flash parts. This allows for booting using the lr/bo booters.

setpci — Set PCI Value

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 with the PCI library. The `setpci` utility may also be used to read a single location in PCI space. The following parameters are included:

`<bus>` = PCI Bus Number 0..255

`<dev>` = PCI Device Number 0..32

`<func>` = PCI Function Number 0..7

`<offset>` = Offset value (i.e. 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 7 0 0x10 d

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

PCI Value.....0x7feff000 (dword) READ

PCI Bus.........0x00
PCI Device......0x07
```

```
PCI Function....0x00
PCI Offset....0x0010
$ setpci 0 7 0 0x10 d 0x1234500

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

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

PCI Bus.........0x00
PCI Device......0x07
PCI Function....0x00
PCI Offset....0x0010
$ setpci 0 7 0 0x10 d

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

PCI Value.....0x01234000 (dword) READ

PCI Bus.........0x00
PCI Device......0x07
PCI Function....0x00
PCI Offset....0x0010
```

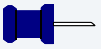
Intel Work Bench Daemons

The Intel® Developer Workbench daemons for the RadiSys ENP-3511 are located in the following directory:

`MWOS/OS9000/ARMV4/PORTS/IXP1200/CMDS`

The following daemons allow the Intel® Developer Workbench to interact with the RadiSys ENP-3511:

`ixp_engine`	System state deamon used to setup an OS-9 to `ixp_serv` interrupt interface.
`ixp_serv`	User state thread-based deamon that communicates with the IXP1200 Developer Workbench to provide microcode load and debug support.



Note

Non-CSL versions of the daemons are located in the following directory:

`mwos\os9000\ARMV4\PORTS\IXP1200\CMDS\NOCSL`

Dependencies

Note

There are incompatibilities between version 1.0 and 1.2 of the Intel® Developer Workbench. To deal with these incompatibilities a 1.0 version compatible daemon, `ixp_serv1_0`, is included. It is used in the same manner as `ixp_serv` but works with version 1.0 of the Intel® Developer Workbench.

The `ixp_serv` and `ixp_engine` daemons provide communication between the Windows host and the target system via RPC over an Ethernet interface. To use the daemons you must perform minimal configuration on both the RadiSys ENP-3511 and on the Windows® host system.

On the RadiSys ENP-3511 your boot file (`os9kboot`) must include a properly configured Ethernet interface.

For More Information

Refer to the **Creating the ROM Image** section for more information on how to create a ROM image.

At boot time, ensure that the network interface is running, along with the RPC portmapper service. The example in this section shows one variation of daemon startup.

From the Windows host system, you must start the Intel® Developer Workbench daemons. After loading a project, set it to the hardware option. Choose Ethernet as your means of communication and set the IP address of the RadiSys ENP-3511.

For More Information

Refer to Appendix B: Running OS-9 and the Intel® Developer Workbench for more information on the Intel® Developer Workbench.

uclo — Load Microcode Object File

SYNTAX

```
uclo [<options>] [<microcode object files>]
```

OPTIONS

-?	Display help.

DESCRIPTION

The uclo utility loads a microcode object file into the IXP1200 Network Processor's microengines. The microcode object file must be in UOF format. (See the ***Radisys Network Processor Development Tool User's Guide*** for more information about generating and using UOF files.) The uclo utility causes the microengines to be initialized, but it does not start the IXP1200 Network Processor's microengines; this must be done by an application, driver, or other StrongARM code.

Example Daemon Start Up

```
OS-9 Bootstrap for the ARM (Edition 65)

Now trying to Override autobooters.

Press the spacebar for a booter menu

Now trying to Copy embedded OS-9 to RAM and boot.
Now searching memory ($00040000 - $001fffff) for an OS-9
Kernel...

An OS-9 kernel was found at $00040000
A valid OS-9 bootfile was found.
$ mbinstall
$ ipstart
$ portmap &
+3
$ ixp_engine &
+5
$ ixp_serv &
+6
$
```

Appendix A: Board Specific Modules

This chapter describes the modules specifically written for the target board. It includes the following sections:

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

Low-Level System Modules

The following low-level system modules are tailored specifically for the RadiSys ENP-3511. The functionality of many of these modules can be altered through changes to the configuration data module (`cnfgdata`).These modules are located in the following directory:

`MWOS/OS9000/ARMV4/PORTS/IXP1200/CMDS/BOOTOBJS/ROM`

`armtimr`	Provides low-level timer services via time base register.
`cnfgdata`	Contains the low-level configuration data.
`cnfgfunc`	Provides access services to the `cnfgdata` data.
`commcnfg`	Inits communication port defined in `cnfgdata`.
`conscnfg`	Inits console port defined in `cnfgdata`.
`ioixp1200`	Provides low-level serial services via the IXP1200 serial unit.
`llne2000`	Provides low-level Ethernet services for NE2k compat PCI cards.
`lle509_pci`	Provides low-level Ethernet services via 3COM PCI cards.
`llpro100`	Provides low-level Ethernet services via Intel 825xx PCI cards.
`llfa3111`	Provides low-level Ethernet service via Netgear `fa3111`
`ll21040_mii`	Provides low-level Ethernet services via DEC/compat 21xxx PCI cards
`portmenu`	Inits booters defined in the `cnfgdata`.
`romcore`	Provides board-specific initialization code.

`usedebug`	Initializes low-level debug interface to RomBug, SNDP, or none.
`ioixp1200`	Provides low level serial access to the 1200's UART.
`initext`	initialized host-to-PCI bridge, 21555 PCI-to-PCI bridge, and PCI bus devices

High-Level System Modules

The following OS-9 system modules are tailored specifically for the RadiSys ENP-3511. Unless otherwise specified, each module is located in a file of the same name in the following directory:

`MWOS/OS9000/ARMV4/PORTS/IXP1200/CMDS/BOOTOBJS`

CPU Support Modules

These files are located in the following directory:

`MWOS/OS9000/ARMV4/CMDS/BOOTOBJS`

`kernel`	Provides all basic services for the OS-9 system.
`cache`	Provides cache control for the CPU cache hardware. The `cache` module is in the file `cach1100`.
`fpu`	Provides software emulation for floating point instructions.
`ssm`	System Security Module—provides support for the Memory Management Unit (MMU) on the CPU.
`vectors`	Provides interrupt service entry and exit code. The `vectors` module is found in the file `vect110`.

System Configuration Module

The system configuration modules are located in the following directory:

`MWOS/OS9000/ARMV4/PORTS/IXP1200/CMDS/BOOTOBJS/INITS`

`dd`	Descriptor module with high level system initialization information.
`nodisk`	Descriptor module with high level system initialization information, but used in a diskless system.
`configurer`	Descriptor module with high level system (generated by the Wizard)

Interrupt Controller Support

The interrupt controller support module provides an extension to the vectors module by mapping the single interrupt generated by an interrupt controller into a range of pseudo vectors. The pseudo vectors are recognized by OS-9 as extensions to the base CPU exception vectors. Refer to the **Port Specific Utilities** section for more information.

`irqixp1200`	P2module that provides interrupt acknowledge and dispatching support for the SA1200's pic (vector range 0x40-0x7d).

Real Time Clock

`rtcixp1200`	Driver that provides OS-9 access to the SA1200's on-board real time clock.

Ticker

`tkarm`	Driver that provides the system ticker based on the ENP-3510's PCI timer.
`hcsub`	Subroutine module that provides a high speed timer interface used by the HawkEye Profiler

Generic I/O Support Modules (File Managers)

The generic I/O support modules are located in the following directory:

`MWOS/OS9000/ARMV4/CMDS/BOOTOBJS`

`ioman`	Provides generic I/O support for all I/O device types.
`scf`	Provides generic character device management functions.
`rbf`	Provides generic block device management functions for the OS-9 format.
`pcf`	Provides generic block device management functions for MS-DOS FAT format.
`spf`	Provides generic protocol device management function support.
`pipeman`	Provides a memory FIFO buffer for communication.

Pipe Descriptor

The pipe descriptor is located in the following directory:

`MWOS/OS9000/ARMV4/PORTS/IXP1200/CMDS/BOOTOBJS/DESC`

`pipe`	Pipeman descriptor that provides a RAM-based FIFO, which can be used for process communication.

RAM Disk Support

The pipe descriptor is located in the following directory:

`MWOS/OS9000/ARMV4/CMDS/BOOTOBJS/`

`ram`	RBF driver that provides a RAM-based virtual block device

RAM Descriptors

The RAM descriptors are located in the following directory:

`MWOS/OS9000/ARMV4/PORTS/IXP1200/CMDS/BOOTOBJS/DESC/RAM`

`r0`	RBF descriptor that provides access to a RAM disk.
`r0.dd`	RBF descriptor that provides access to a ram disk—with module name dd (for use as the default device).
`r1`	RBF descriptor that provides access to a 4Meg RAM disk for Flash burning.

Serial and Console Devices

`scixp1200`	SCF driver that provides serial support the SA1200's internal UART.

Descriptors for use with sc1100

`term1/t1`	Descriptor modules for use with `scixp1200` ENP-3510 Board header: J20 Default Baud Rate: 38400 Default Parity: None Default Data Bits: 8 Default Handshake: XON/XOFF
`scllio`	SCF driver that provides serial support via the polled low-level serial driver.

Descriptors for use with scllio

The `scllio` descriptors are located in the following directory:

`mwos\os9000\ARMV4\PORTS\IXP1200\CMDS\BOOTOBJS\DESC\SCLLIO`

`vcons/term`	Descriptor modules for use with `scllio` in conjunction with a low-level serial driver. Port configuration and set up follows that which is configured in `cnfgdata` for the console port. It is possible for `scllio` to communicate with a true low-level serial device driver like `io1100`, or with an emulated serial interface provided by `iovcons`.

For More Information

Refer to the ***OS-9 Porting Guide*** and the ***OS-9 Device Descriptor and Configuration Module Reference*** for more information.

SPF Device Support

PCI Support for NE2000 Compatibility

The NE2000 support module is located in the following directory:

`MWOS/OS9000/ARMV4/PORTS/IXP1200/CMDS/BOOTOBJS/SPF`

`spne2000`	SPF driver to support PCI NE2000 Ethernet cards.

The following cards are supported: RealTek RTL-8029, Winbond 89C940, Winbond w89C940, Compex RL2000, KTI ET32P2, NetVin NV500SC, Via 82C926, SureCom NE34, Holtek HT80232, Holtek HT80229.

spne2000 Descriptors

This descriptor file is located in the following directory:
`MWOS/OS9000/ARMV4/PORTS/IXP1200/CMDS/BOOTOBJS/SPF`

`spne0`	SPF descriptor module for use with PCI.

PCI Support for 3COM Ethernet cards

The 3COM support module is located in the following directory:

`MWOS/OS9000/ARMV4/PORTS/IXP1200/CMDS/BOOTOBJS/SPF`

`spe509`	SPF driver to support ethernet for a 3COM PCI cards.

The following cards are supported: 5900, 9001, 3C900-TPO, 3C905-TX, 3C905-T4, 3CSOHO100-TX, 3C905B-TX,3C900B-TPO 3C900B-CMB, 9058, 9006, 900A, 905A, 9200, 9800, 9805

spe509 Descriptors

These descriptor files are located in the following directory:
`MWOS/OS9000/ARMV4/PORTS/IXP1200/CMDS/BOOTOBJS/SPF`

`spe30`	SPF descriptor module for use with default PCI slot 0.
`spe31`	SPF descriptor module for use with default PCI slot 1.
`spe32`	SPF descriptor module for use with default PCI slot 2.
`spe33`	SPF descriptor module for use with default PCI slot 3.

PCI Support for Intel Ethernet Pro cards

The Intel Ethernet Pro support module is located in the following directory:

`MWOS/OS9000/ARMV4/PORTS/IXP1200/CMDS/BOOTOBJS/SPF`

`sppro100`	SPF driver to support ethernet for a Intel Ethernet Pro PCI cards.

The following cards are supported: 82557, 82559ER, 1029, 1030

sppro100 Descriptors

These descriptor files are located in the following directory:
`MWOS/OS9000/ARMV4/PORTS/IXP1200/CMDS/BOOTOBJS/SPF`

`sppr0`	SPF descriptor module for use with default PCI slot 0.
`sppr1`	SPF descriptor module for use with default PCI slot 1.
`sppr2`	SPF descriptor module for use with default PCI slot 2.
`sppr3`	SPF descriptor module for use with default PCI slot 3.

PCI Support for DEC 211xx Ethernet cards

The DEC support module is located in the following directory:

`MWOS/OS9000/ARMV4/PORTS/IXP1200/CMDS/BOOTOBJS/SPF`

`sp21140`	SPF driver to support ethernet for a Intel Ethernet PRO PCI cards.
decv	sp21140 utility that shows the state of the Ethernet chip

The following cards are supported: DEC21143, DEC21140, DEC2104, Netgear FA310.

sp21140 Descriptors

These descriptor files are located in the following directory:
`MWOS/OS9000/ARMV4/PORTS/IXP1200/CMDS/BOOTOBJS/SPF`

`spde0`	SPF descriptor module for use with default PCI slot 0.
`spde1`	SPF descriptor module for use with default PCI slot 1.
`spde2`	SPF descriptor module for use with default PCI slot 2.
`spde3`	SPF descriptor module for use with default PCI slot 3.

PCI Support for National DP83815 Ethernet Card

The National DP83815 support module is located in the following directory:

`MWOS/OS9000/ARMV4/PORTS/IXP1200/CMDS/BOOTOBJS/SPF`

`spfa311`	SPF driver to support Ethernet for Netgear FA311/312 83815 clones.

spfa311 Descriptors

These descriptor files are located in the following directory:
`MWOS/OS9000/ARMV4/PORTS/IXP1200/CMDS/BOOTOBJS/SPF`

`spfa0`	SPF descriptor module for use with default PCI slot 0.
`spfa1`	SPF descriptor module for use with default PCI slot 1.
`spfa2`	SPF descriptor module for use with default PCI slot 2.
`spfa3`	SPF descriptor module for use with default PCI slot 3.

Support for Communication Across the PCI Backplane

The PCI Backplane support module is located in the following directory:

`MWOS/OS9000/ARMV4/PORTS/IXP1200/CMDS/BOOTOBJS/SPF`

The following boards are supported as targets:

- ENP-3511
- ENP-2505
- Intel Ethernet Eval

`spixhpc`	is the SPF driver to support backplane communications as a target. The descriptor files for `spixhpc` are located in the following directory: `MWOS/OS9000/ARMV4/PORTS/IXP1200/CMDS/BOOTOBJS/SPF` `sppc0` is the SPF descriptor module for use as the target. `sppch` is the SPF descriptor module for use as the host.
`spixhpch`	is the SPF driver to support backplane communications as a host.

Network Configuration Modules

These files are located in the following directory:
`MWOS/OS9000/ARMV4/PORTS/IXP1200/CMDS/BOOTOBJS/SPF`

`inetdb`

`inetdb2`

`rpcdb`

Common System Modules List

The following low-level system modules provide generic services for OS9000 Modular ROM. They are located in the following directory:

`MWOS/OS9000/ARMV4/CMDS/BOOTOBJS/ROM`

`bootsys`	Provides booter registration services.
`console`	Provides console services.
`dbgentry`	Inits debugger entry point for system use.
`dbgserve`	Provides debugger services.
`excption`	Provides low-level exception services.
`flshcach`	Provides low-level cache management services.
`hlproto`	Provides user level code access to `protoman`.
`llbootp`	Provides bootp services.
`llip`	Provides low-level IP services.
`llslip`	Provides low-level SLIP services.
`lltcp`	Provides low-level TCP services.
`lludp`	Provides low-level UDP services.
`llkermit`	Provides a booter that uses kermit protocol.
`notify`	Provides state change information for use with LL and HL drivers.
`override`	Provides a booter which allows choice between menu and auto booters.
`parser`	Provides argument parsing services.
`pcman`	Provides a booter that reads MS-DOS file system.

`protoman`	Provides a protocol management module.
`restart`	Provides a booter that causes a soft reboot of system.
`romboot`	Provides a booter that allows booting from ROM.
`rombreak`	Provides a booter that calls the installed debugger.
`rombug`	Provides a low-level system debugger.
`sndp`	Provides low-level system debug protocol.
`srecord`	Provides a booter that accepts S-Records.
`swtimer`	Provides timer services via software loops.

For More Information

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

Appendix B: Running OS-9 and the Intel® Developer Workbench

This appendix provides a brief overview of using the Intel® Developer Workbench with an OS-9 system. It includes the following sections:

- **Overview**
- **Intel Developer Workbench**

Overview

The OS-9 ROM image provides an interface with the Intel Core Software Library. This enables loading and debugging of microcode to the Intel® Developer Workbench in a simulated environment or directly on the hardware. The simulated environment has more features than the hardware version and is more appropriate for detailed debugging.

The standard OS-9 image also contains the SPF daemons for Hawk debugging on the StrongARM core, the daemons for using the Profiler, and most standard OS-9 utilities.

Intel Developer Workbench

All RadiSys ENP-3511 Embedded Network Processor boards ship with the Intel® Developer Workbench.

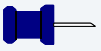
For More Information

The Intel Developer Workbench, which should have been included with your hardware, is also available on CD from the Intel® Literature Center at the following URL:

```
http://apps.intel.com/scripts-order/
viewBasket.asp?Bundle=Bundle&site=developer
```

You can also order by phone at 800-548-4725, 7am to 7pm CST. Please ask for part number CDIXP12DE11. Outside U.S. please allow 2-3 weeks for delivery.

The Enhanced OS-9 for IXP1200 board-level solution provides the necessary daemons to enable integration of the Intel and OS-9 environments. The following procedures describe how to configure the environment and run a sample microcode project.



Note

The following procedures assume that you have completed the steps detailed in **Chapter 1**.

System Configuration

To configure your system to use the Intel® Developer Workbench, complete the following steps:

Step 1. Install version 1.2 of the Intel® Developer Workbench. Installation requires a key, available from the Intel website at the following URL:

```
http://developer.intel.com/design/network/products/npfamily/ixp1200.htm
```

Step 2. Run the sample project. In the Hyperterminal window, the following messages display as the RadiSys ENP-3511 is booted up:

```
OS-9 Bootstrap for the ARM (Edition 65)

Now trying to Override autobooters.

Press the spacebar for a booter menu

Now trying to Copy embedded OS-9 to RAM and boot.
Now searching memory ($00040000 - $001fffff) for an OS-9 Kernel...

An OS-9 kernel was found at $00040000

A valid OS-9 bootfile was found.
```

Step 3. Start networking by typing the following command at the OS-9 prompt: `ipstart &`.

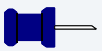
Step 4. Start RPC by typing the following command at the OS-9 prompt: `portmap &`.

Step 5. Start the first Intel® Developer Workbench daemon by typing the following command at the OS-9 prompt: `ixp_engine &`.

Step 6. Start the second Intel® Developer Workbench daemon by typing the following command at the OS-9 prompt: `ixp_serv &`.

Step 7. Configure the Ethernet settings and check the network settings by typing the following two commands at the OS-9 prompt:

```
$ ifconfig enet0 172.16.1.236
$ netstat -in
Name  Mtu   Network     Address              Ipkts Ierrs    Opkts Oerrs  Coll
lo0   1536  <Link>                             4     0        4     0      0
lo0   1536  127         127.0.0.1              4     0        4     0      0
enet0 1500  <Link>      00.02.B3.07.12.01    267     0        0     0      0
enet0 1500  172.16      172.16.1.236         267     0        0     0      0
```



Note

The IP address used above is an example only. You must get your specific network information from your network administrator.

Running the Sample Project

The following example requires the Hawk and Profiler daemons to be running on the RadiSys ENP-3511.

Step 1. Start the Hawk daemon by typing the following command at the OS-9 prompt (in the Hyperterminal window):

```
spfndpd &
```

Step 2. Start Profiler daemon by typing the following command at the OS-9 prompt:

```
spfnppd &
```

Step 3. From the Windows host system, select `Start -> Programs -> Intel IXP1200 -> Developer Workbench`. The Intel® Developer Workbench opens in the foreground.

Step 4. From the Intel® Developer Workbench select `File -> Open Project`. The standard Windows file search box is displayed.

Step 5. Browse to and open the following file:

```
IXP1200\Microcode\examples\bit_swiz.dwp
```

The project is opened and the file tree is visible on the right (source macro script).

Step 6. Select `Debug -> Hardware`. A check box goes to the hardware menu.

Step 7. Select `Hardware -> Options`. The **Hardware Options** screen is displayed.

Step 8. Click on **Connect via Ethernet** and enter your target IP address.

Step 9. Click `OK`.

Step 10. Click on the **Bug** picture on the right side of the screen. The Intel® Developer Workbench screen changes, and the MicroEngine window opens at the bottom. You may have to click on **spool**.

Step 11. Expand the MicroEngine 0 by clicking +. Thread 0-0 through Thread 3-0 will be exposed.

Step 12. Click the green traffic sign to load and start executing the microcode. The arrow in the MicroEngine window will advance as code runs on different engines.

Intel® Developer Workbench Interface Notes

- The button that resembles steps (labeled **HOP)** will single step after stopping.
- Double clicking on code can bring it to the foreground, and arrows will move through the code when it is being executed.
- Right clicking and holding brings up a breakpoint window. This can also be done from the debugger pull-down menu.
- This procedure can be run in the SIMULATOR by not choosing the two hardware options WB4/WB5. This gives you a practice run.

Appendix C: Ethernet Router Application

Your ENP-3511 board may have been configured to include Ethernet router demonstration software. This is a factory demonstration that will stop running after eight hours. The following section explains how to re-create the bundle demonstration without the time limitation.

Using the Ethernet Router Application

Complete the following steps to re-create the Ethernet Router Bundle demonstration without a time limitation. Before you begin, however, make sure you have the following add-on products installed on your development system: EMANATE/Lite 15.3 SNMP for OS-9, IP Infusion OSPFv2 v1.1 for OS-9, IP Infusion RIPv1v2 v1.1 for OS-9, IP Infusion ZebOS Daemon v1.1 for OS-9, IXP1200 Forwarding Table Manager for OS-9, and Enhanced OS-9 for IXP1200.

Step 1. From the opening Configuration Wizard screen, under **Port Selection**, select `Radisys ENP-3511`. Under **Configuration Name**, type `ethroute_3511`. Click `OK`.

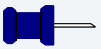
Step 2. This step is showing you which check boxes were pre-selected for you to include the proper routing software. After the Wizard has started, select `Configure -> Bootfile -> Network Configuration` from the menu.

Step 3. From the **Network Configuration** dialog, select the **SoftStax Options** tab. You should see the following check boxes selected: snmp, Zebos OSPF, Zebos RIP, FTM, and Routing Demo. Click `Cancel`.

Step 4. Select `Configure -> Build Image`. From the **Master Builder** dialog, click `Build`. After the bootfile has finished building, you can transfer the image to the ENP-3511 and program it into Flash.

Setting up the Target System

To set up the demonstration system to run the Ethernet Router Application, complete the following steps:



Note

Be sure the board is powered off before you begin these steps.

Step 1. If you are connecting to a hub:

- Connect a straight-through Ethernet cable to the Intel Ethernet Pro port located on the front of the board. In addition, connect reverse Ethernet cables to each port used on the transition module.

If you are not connecting to a hub:

- Use crossover cables to make the connection.

WARNING

Do not connect any of the Ethernet ports (IXP1200 microcode ports) on the transition module to the same network that connects the Intel® Ethernet Pro port.

Step 2. Connect the COM port to your PC, using an RS-232 serial cable.

For More Information

For more information on how to configure and establish a connection through the serial port, refer to the board guide for your respective processor, included with this CD.

Step 3. Apply power to the board.

Note

You may need to press the reset button in order for the board to boot properly.

Upon booting the board, you should see the following similar message on your terminal:

```
WARNING: IXF440 MAC Address Initialzation failed. sts =  130
Press space bar to stop auto-boot...
10
OS-9 Bootstrap for the ARM (Edition 67)
Now trying to Override autobooters.
Press the spacebar for a booter menu
Now trying to Copy embedded OS-9 to RAM and boot.
Now searching memory ($00240000 - $007fffff) for an OS-9 Kernel...
An OS-9 kernel was found at $00240000
A valid OS-9 bootfile was found.
+3
$
```

Note

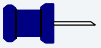
If you would like to adjust the time it takes in the Boot Manager to begin booting OS-9, refer to the **Optional Procedures** section of Chapter one.

At this point OS-9 should have booted and the networking components should be enabled.

To use the Intel® Ethernet Pro port, contact your network administrator and get a valid IP address & Subnet mask for your network. To configure the Intel® Ethernet Pro network interface type the following at the OS-9 prompt:

```
ifconfig enet0 <IP address> netmask <subnet mask>
binding /sppr0/enet
```

You should now be able to use the OS-9 networking utilities to communicate across your network.



Note

If you create a custom ROM image with the Configuration Wizard and specify valid network addresses for use on your network you will not have to use the ifconfig utility to configure the network interface.