OS-9 for Intel® IXM1200 Network Processor Base Card 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
December 2001

Copyright and publication information

Disclaimer

Reproduction notice

Table of Contents

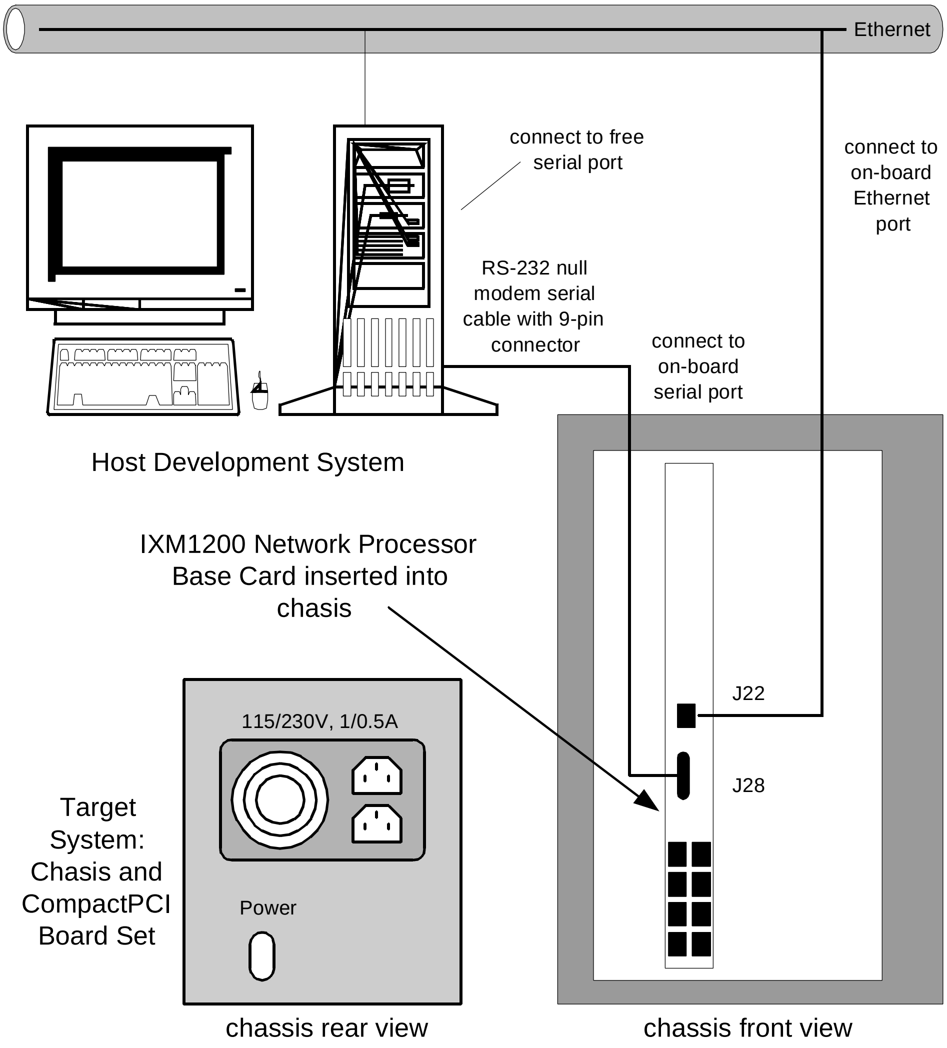
Chapter 1: Installing and Configuring OS-9

This chapter describes installing and configuring OS-9 on the Intel® IXM1200 Network Processor Base Card. It includes the following sections:

Development Environment Overview

Figure 1-1 shows a typical development environment for the Intel IXM1200 Network Processor Base Card. The components shown include the minimum required to enable OS-9 to run on the Intel IXM1200 Network Processor Base Card.

Figure 1-1 IXM1200 Development Environment


Ethernet
connect to free serial port
connect to on-board Ethernet port
RS-232 null modem serial cable with 9-pin connector
connect to on-board serial port
Host Development System
IXM1200 Network Processor Base Card inserted into chasis
J22
J28
115/230V, 1/0.5A
Power
Target System: Chasis and CompactPCI Board Set
chassis rear view
chassis front view

Requirements and Compatibility

Note

Before you begin, install the ***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, or Windows 2000
- terminal emulation program

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

Target Hardware Requirements

Your reference board requires the following hardware:

- enclosure or chassis with power supply and backplane
- one OS-9 Flash part
- a RS-232 null modem serial cable with 9-pin connectors
- an Ethernet network card
- access to an Ethernet network

Target Hardware Setup

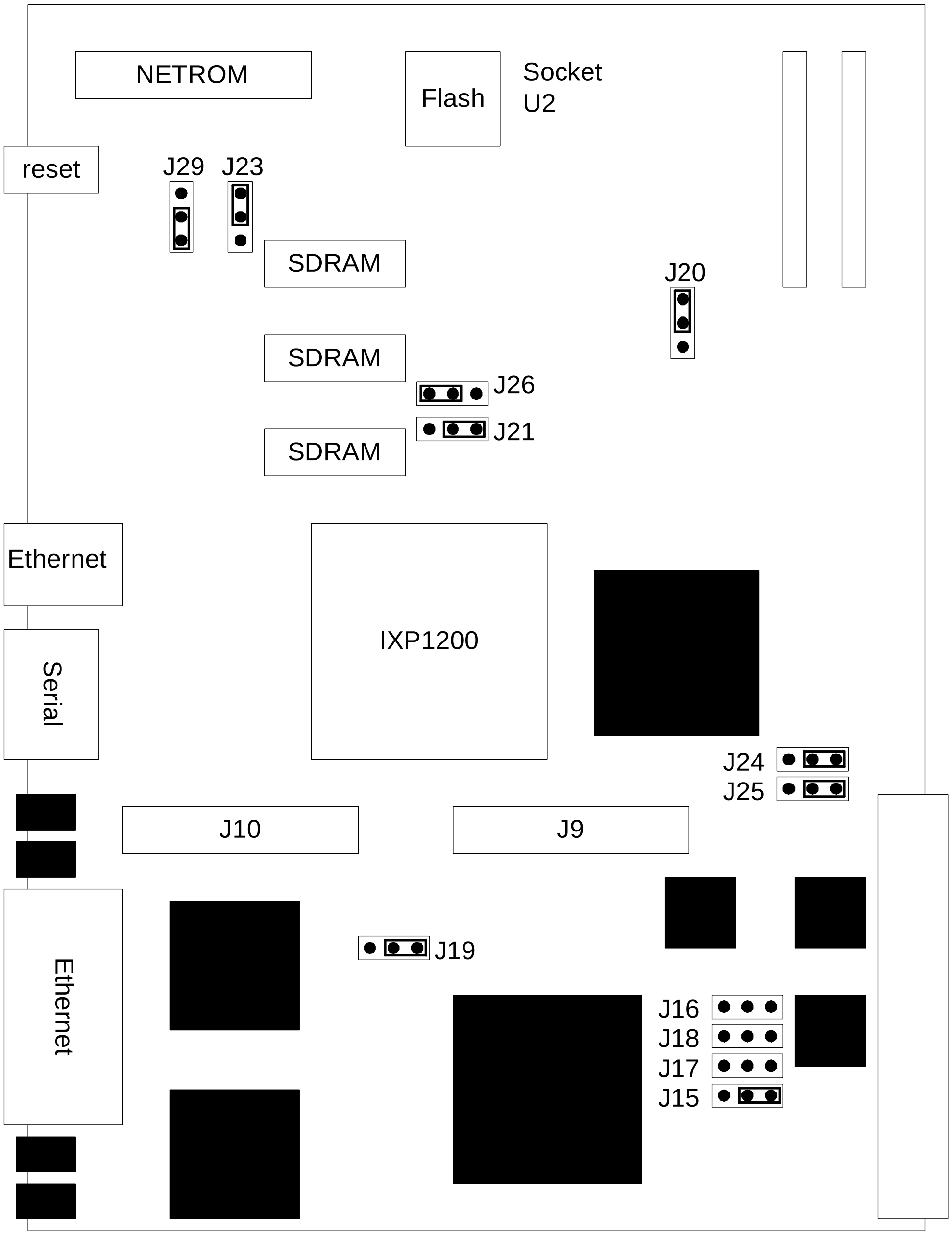
Installing the Flash Part

Configuring your reference board consists of installing the Flash part included in your ***Enhanced OS-9 for IXP1200*** product. The Flash part includes a coreboot image that has been pre-configured to get your board up and running quickly. To install the Flash part, complete the following steps:

Step 1. With your target system powered down, remove the IXM1200 Network Processor Base Card from the chassis. Lay the board on a flat, static-free surface.

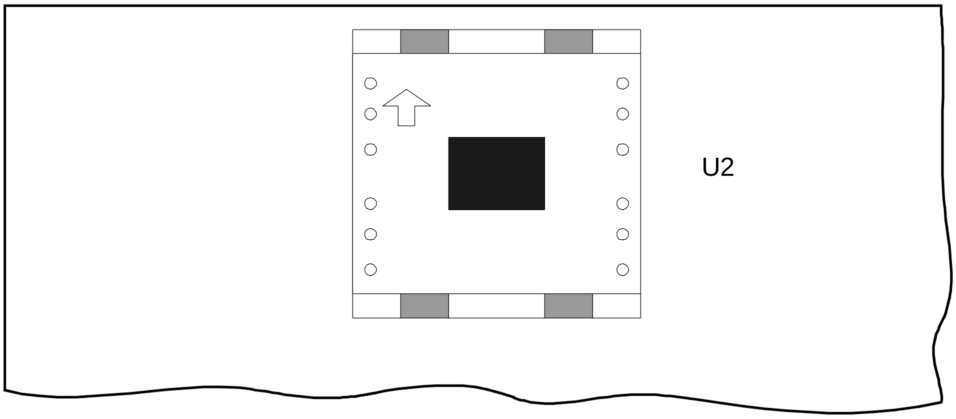
Step 2. Remove the factory Flash part from socket U2, shown in **Figure 1-2**, by using your thumb to firmly push the stainless steel Flash part gate in the direction of the arrow, shown in **Figure 1-3**.

Figure 1-2 IXM1200 Network Processor Base Card



An arrow is lightly etched into the Flash part gate as shown in **Figure 1-3**. Remove the factory Flash part from the socket and store it in a safe place.

Figure 1-3 Flash Part Gate



Step 3. Insert the OS-9 Flash part into socket U2. Use the OS-9 Flash part supplied with your software package.

Step 4. Replace the Flash part gate and slide back into place.

WARNING

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

Note

If you need to reprogram the Flash devices or create new Flash devices, see the **Optional Procedures** section.

Jumper Settings

All jumper settings can remain as shipped from the factory. The jumpers are used only in conjunction with the factory Flash part—not the OS-9 Flash part.

Connecting the Target to the Host

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. For a description see **Figure 1-1**.

Step 3. Connect the target system to the host system using an RS-232 null modem serial cable with 9-pin connectors. For a description see **Figure 1-1**.

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

Step 5. Open Hyperterminal and enter a name for your session.

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

Step 7. In the **Phone Number** dialog, go to the **Connect Using** box, and select the communications port to be used to connect to the reference board.

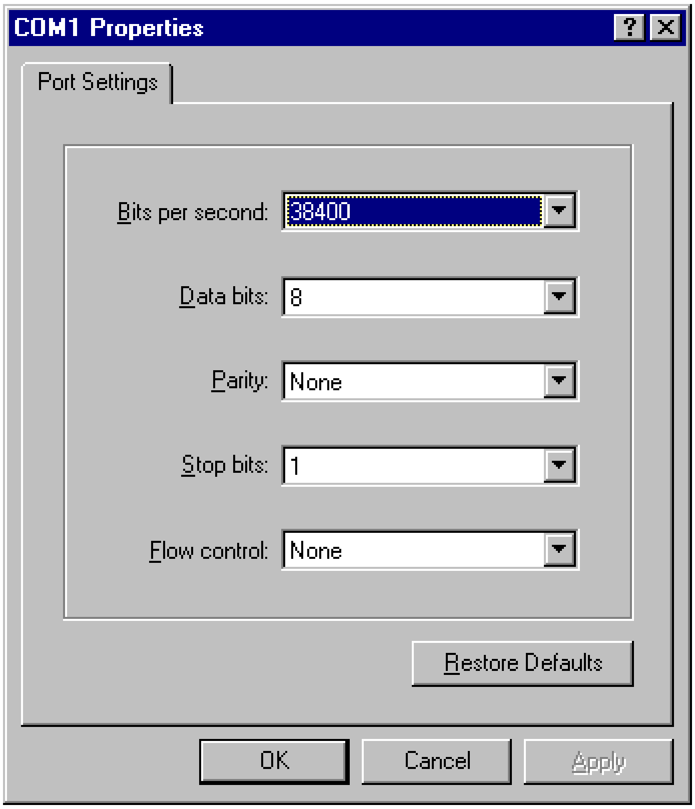
The port you select must be the same port that you inserted the actual cable into.

Step 8. Click `OK`.

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

```
Bits per second = 38400

Data Bits = 8

Parity = None

Stop bits = 1

Flow control = None
```

Figure 1-4 Port Settings


COM1 Properties

Port Settings

Bits per second: 38400

Data bits: 8

Parity: None

Stop bits: 1

Flow control: None

Step 10. Click `OK.` A connection should be established.

Note

If the word *connected* does not appear in the lower left corner of the window, click `Call` -> `Connect` to establish a connection.

Step 11. Apply power to the board. 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.

OS-9 ROM Image Overview

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.

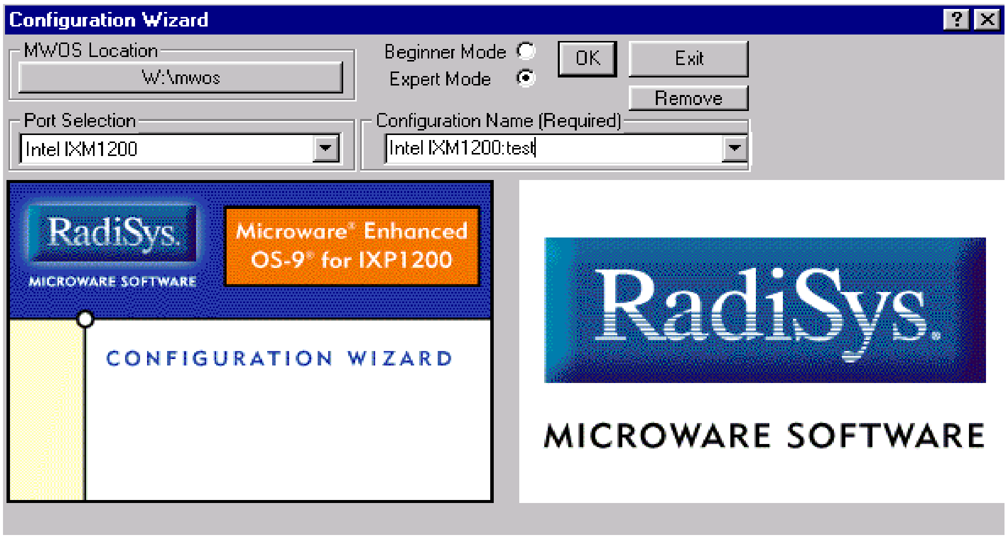
Microware provides a Configuration Wizard to create a coreboot image, a bootfile image, or an entire OS-9 ROM Image. The wizard can also be used to modify an existing image. The Configuration Wizard is automatically installed on your host PC during the installation process.

Using the Configuration Wizard

To use the Configuration Wizard, perform the following steps:

Step 1. From the Windows desktop, select `Start -> Programs -> RadiSys -> Enhanced OS-9 for IXP1200 <ver> -> Configuration Wizard`. You should see the following opening screen:

Figure 1-5 StrongARM Configuration Wizard



Step 2. Select the path where the MWOS directory structure can be located by clicking the **MWOS location** button.

Step 3. Select the target board from the **Port Selection** pull-down menu.

Step 4. Select a name for your configuration in the **Configuration Name** field. Your settings will be saved for future use.

Step 5. 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.

Building a Bootfile for the IXM1200 Base Card

The OS-9 Flash parts shipped with Enhanced OS-9 for IXP1200 3.0.1 include the minimum software required to boot the target to an OS-9 prompt. The following procedure describes using the configuration wizard to modify the bootfile and add functionality to your system.

Two of the most common modifications are described below. These include adding networking functionality and enabling a host/target connection via Hawk, the Microware integrated development environment.

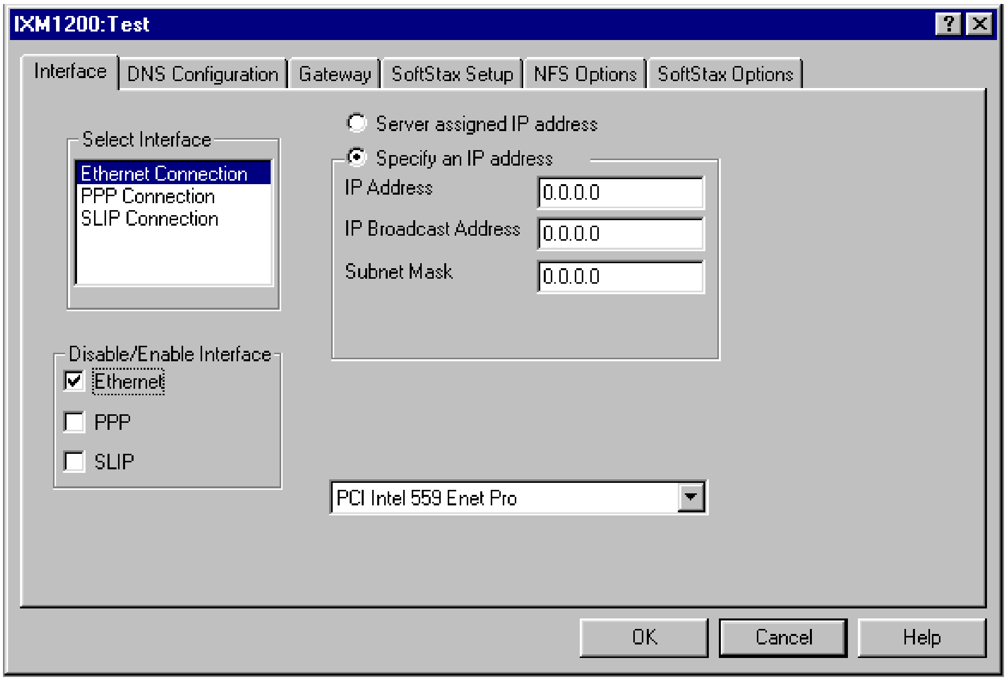
To configure your system for networking and add Hawk functionality, complete the following steps:

Note

Advanced configuration procedures are described in the **Optional Procedures** section.

Step 1. From the wizard's main configuration window, select `Configure -> Bootfile -> NetWork Configuration`. The following dialog window should appear:

Figure 1-6 Network Configuration—Interface Tab



Select Interface: Ethernet Connection (selected), PPP Connection, SLIP Connection

( ) Server assigned IP address
(●) Specify an IP address

IP Address: 0.0.0.0
IP Broadcast Address: 0.0.0.0
Subnet Mask: 0.0.0.0

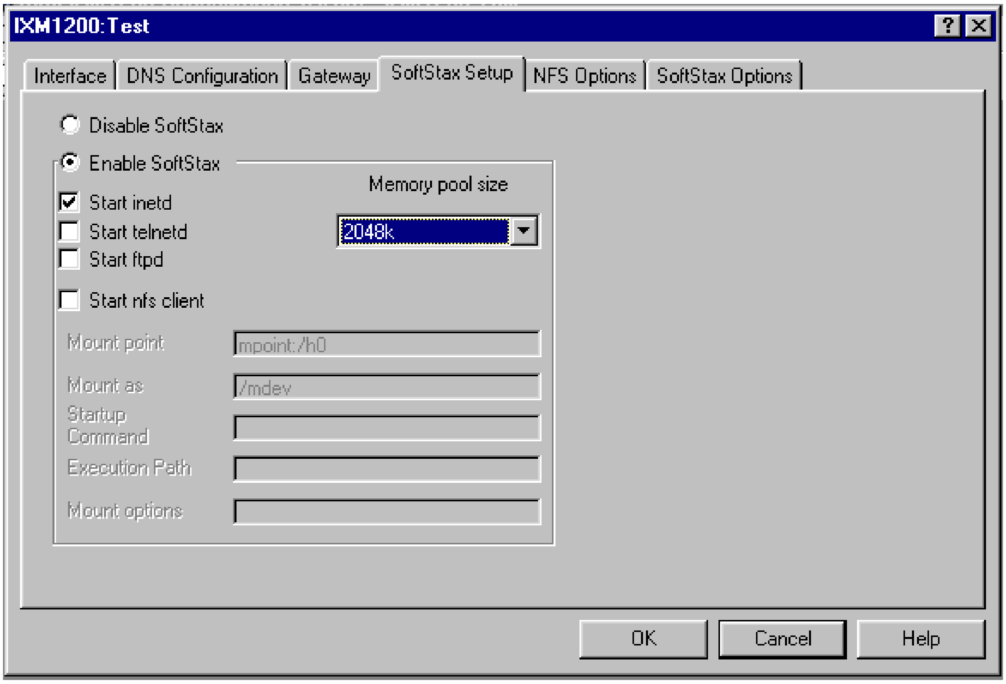
Disable/Enable Interface:
✓ Ethernet
☐ PPP
☐ SLIP

PCI Intel 559 Enet Pro

Step 2. Configure your system as shown in Figure 1-6. 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 following dialog window should appear:

Figure 1-7 Network Configuration—SoftStax Setup Tab



( ) Disable SoftStax
(●) Enable SoftStax

[✓] Start inetd
[ ] Start telnetd
[ ] Start ftpd
[ ] Start nfs client

Memory pool size: 2048k

Mount point: mpoint:/h0
Mount as: /mdev
Startup Command:
Execution Path:
Mount options:

Step 4. Configure your system as shown in **Figure 1-7**.

Step 5. Leave the other **Network Configuration** options at the default settings. Click `OK`.

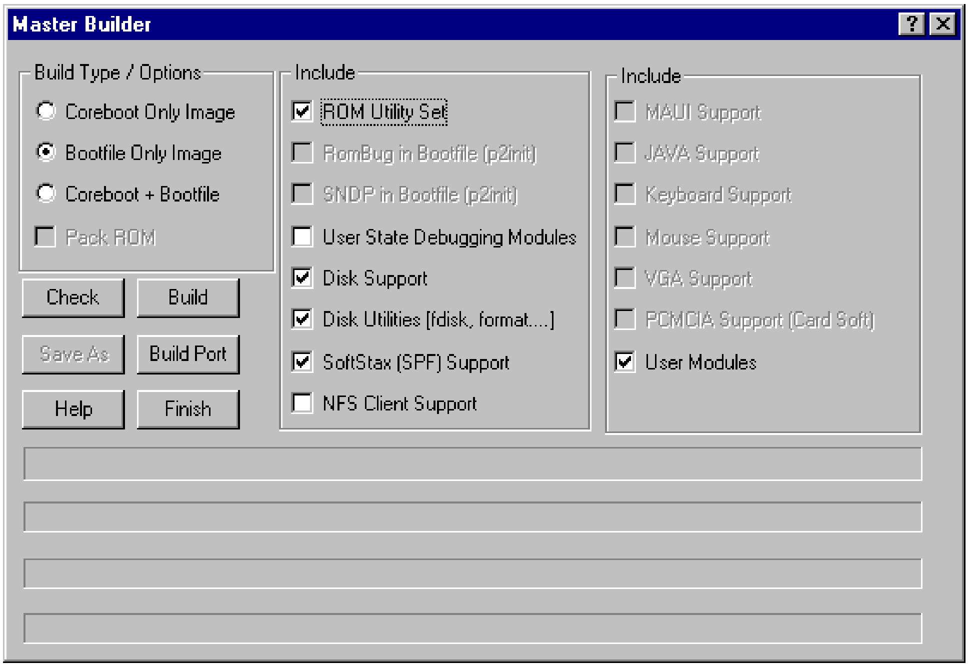
Note

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

Step 6. Select `Configure -> Bootfile -> Disk Configuration`. From the **Init Options** tab, select `/dd` in the **Initial Device Name** field. This automatically enables networking at system start up.

Step 7. Select `Configure -> Build Image`. The **Master Builder** dialog window appears.

Figure 1-8 Master Builder Dialog Window



Master Builder

Build Type / Options
- ○ Coreboot Only Image
- [✓ ●] Bootfile Only Image
- ○ Coreboot + Bootfile
- ☐ Pack ROM

Include
- [✓] ROM Utility Set
- ☐ RomBug in Bootfile (p2init)
- ☐ SNDP in Bootfile (p2init)
- ☐ User State Debugging Modules
- [✓] Disk Support
- [✓] Disk Utilities [fdisk, format....]
- [✓] SoftStax (SPF) Support
- ☐ NFS Client Support

Include
- ☐ MAUI Support
- ☐ JAVA Support
- ☐ Keyboard Support
- ☐ Mouse Support
- ☐ VGA Support
- ☐ PCMCIA Support (Card Soft)
- [✓] User Modules

Step 8. Configure your Master Builder options as shown in **Figure 1-8**.

Step 9. Click `Build` to build a bootfile image. The bootfile image, `os9kboot`, is stored in the following default location:

```
\MWOS\OS9000\ARMV4\PORTS\IXP1200\BOOTS\INSTALL\PORTBOOT\
```

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

Note

The specific contents of your bootfile can vary from system to system depending on hardware capabilities and user requirements.

Transferring the Bootfile to the Target

The following procedures describe transferring the bootfile image from the host system to the target system. The following basic steps are included:

- Manually configure target settings
- Test the Ethernet connection
- Using File Transfer Protocol (FTP) and the OS-9 `pflash` utility to transfer the bootfile image from the host system to the target system

Manually Configure Target

The software shipped on the OS-9 Flash parts includes basic networking functionality. The basic network settings include address fields that must be reconfigured for your particular network. In addition, an area of the target system's Flash must be initialized. Complete the following procedure to manually configure the target.

Step 1. Open the IXM1200 Network Processor Base Card Hyperterminal window on the Windows host desktop.

Step 2. Initialize a RAM Disk to hold temporarily the image to be flashed by typing the following command:

```
$ iniz /r1
```

Step 3. Initialize the networking modules by typing the following command:

```
$ ipstart
```

Step 4. Configure the network interface parameters and add entries to the routing table by typing the following commands:

```
$ ifconfig enet0 x.x.x.x netmask y.y.y.y broadcast y1.y1.y1.y1
```

Note

The placeholders listed above represent the following items:

x.x.x.x is the IP address assigned to the IXM1200 Network Processor Base Card.
y.y.y.y is the subnet mask address.
y1.y1.y1.y1 is the IP Broadcast for your network.
z.z.z.z is the IP Gateway address.

This information must be supplied by your network administrator.

Step 5. Start the internet process server by typing the following command:

```
$ inetd<>>>/nil&
```

Step 6. Change the current directory on the target by typing the following command:

```
$ chd /r1
```

Testing the Ethernet Connection

To confirm that your host and target can communicate over Ethernet, perform the following steps.

Step 1. Open a DOS shell on the host system.

Step 2. At the prompt type the following command:

```
ping <IP address of target>
```

The IP Address is the address you assigned to the target board and is typed without the <> brackets.

If the ping was successful, you will see the following response:

```
Reply from <IP Address>: bytes=xx time =xms TTL= xx
```

If the ping was unsuccessful, you will see the following response:

```
Request timed out.
```

Using FTP and pflash

Complete the following procedure to transfer the bootfile image from the host system to the target system. It is assumed that you have completed the steps described previously in this manual and that the target is connected to an Ethernet network and is booted up.

Step 1. Start a DOS shell on the host system.

Step 2. Change directories to where the bootfile image, `os9kboot`, is located. The default location for this file is in the following directory:

```
\MWOS\OS9000\ARMV4\PORTS\IXP1200\BOOTS\INSTALL\PORTBOOT\
```

Step 3. At the prompt type the following command:

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

Step 4. Log in by typing the following commands at the prompt:

```
user: super
password: user
```

Step 5. At the `ftp>` prompt type the following command:

```
bin
```

This designates binary format.

Step 6. At the `ftp>` prompt type the following command:

```
cd /r1
```

Step 7. At the `ftp>` prompt type the following command:

```
put os9kboot
```

The `os9kboot` file is transferred to the target's RAM disk (`/r1`).

Step 8. Open the IXM1200 Network Processor Base Card Hyperterminal window on the Windows host desktop.

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

```
$ pflash -f=/r1/os9kboot
```

The `os9kboot` file is transferred to the target system's Flash memory. `pflash` is configured to copy the `os9kboot` file to the correct address.

Step 10. Reset the IXM1200 Network Processor Base Card.

For More Information

The pflash utility is documented in the **Port Specific Utilities** section of **Chapter 2**.

The result of this procedure is that the bootfile you created on the Windows host system, `os9kboot`, now resides in the target system's Flash memory and is used every time the XM1200 Network Processor Base Card reboots.

Optional Procedures

Creating a ROM Image (coreboot plus bootfile)

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:

WARNING

Using the OS-9 `pflash` utility to program a new coreboot image into Flash can damage the Flash parts. Any errors made during coreboot image modification or during reburning of the Flash part can result in a non-bootable IXM1200 Board.

Note

These procedures assume that you have already completed the procedures described in the **OS-9 ROM Image Overview** and **Transferring the Bootfile to the Target** 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. Click `OK` to close.

Step 3. Select `Configure -> Bootfile -> Disk Configuration`. Select the **Init Options** tab.

Step 4. Select the **Mshell** radio button.

Step 5. Select the **/dd** radio button.

Step 6. Select the **User** radio button in the **Initial Device Name** field. This will cause the **/dd** radio button to become deselected. This enables you to edit the text in the **Parameters List** text field. Remove the following string from the **Parameter List** text field:

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

WARNING

Failure to remove the text string exactly as shown will corrupt your ROM Image and result in a non-bootable IXM1200 board.

Step 7. Click `OK` to close.

Step 8. Select `Sources -> Port -> User(user.ml)[MASTER]`. Add the following lines above the `../../../../../../ARMV4/CMDS/kermit` entry:

```
* 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
```

Note

It is recommended that you cut and paste this text block directly from the PDF file for this manual. Any errors made while keying in this text block will corrupt your ROM Image and result in a non-bootable IXP1200 board.

Step 9. Save and close the file.

Step 10. Select `Configure -> Build Image`. The Master Builder dialog window appears.

Select the **Coreboot + Bootfile** radio button.

Select the **User State Debugging Modules** check box.

Step 11. Click `Build`.

Step 12. Click `Finish` when the image has been built.

Step 13. Start a DOS shell on the host system.

Step 14. Change directories to where the OS-9 ROM image, `rom`, is located. The default location for this file is in the following directory:

```
\mwos\OS9000\ARMV4\PORTS\IXP1200\BOOTS\INSTALL\PORTBOOT\
```

Step 15. At the prompt type the following command:

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

Step 16. Log in by typing the following commands at the prompt:

```
user: super
password: user
```

Step 17. At the `ftp>` prompt type the following command:

```
bin
```

This designates binary format.

Step 18. At the `ftp>` prompt type the following command:

```
cd /r1
```

Step 19. 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 20. Open the Intel® IXM1200 Network Processor Base Card Hyperterminal window on the Windows host desktop.

Step 21. 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.

Step 22. Reset the target system.

After the target system is reset and booted up, you must choose between a low-level or high-level configuration according to your needs. One of the following two sets of commands must be entered at the OS-9 prompt after reset:

Low-Level Configuration

For a low-level configuration, enter the following commands at the OS-9 prompt:

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

High-Level Configuration

For a high-level configuration, enter the following commands at the OS-9 prompt:

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

To alternate between the two configurations, simply reset the target and enter the appropriate commands at the OS-9 prompt.

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, you have 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.

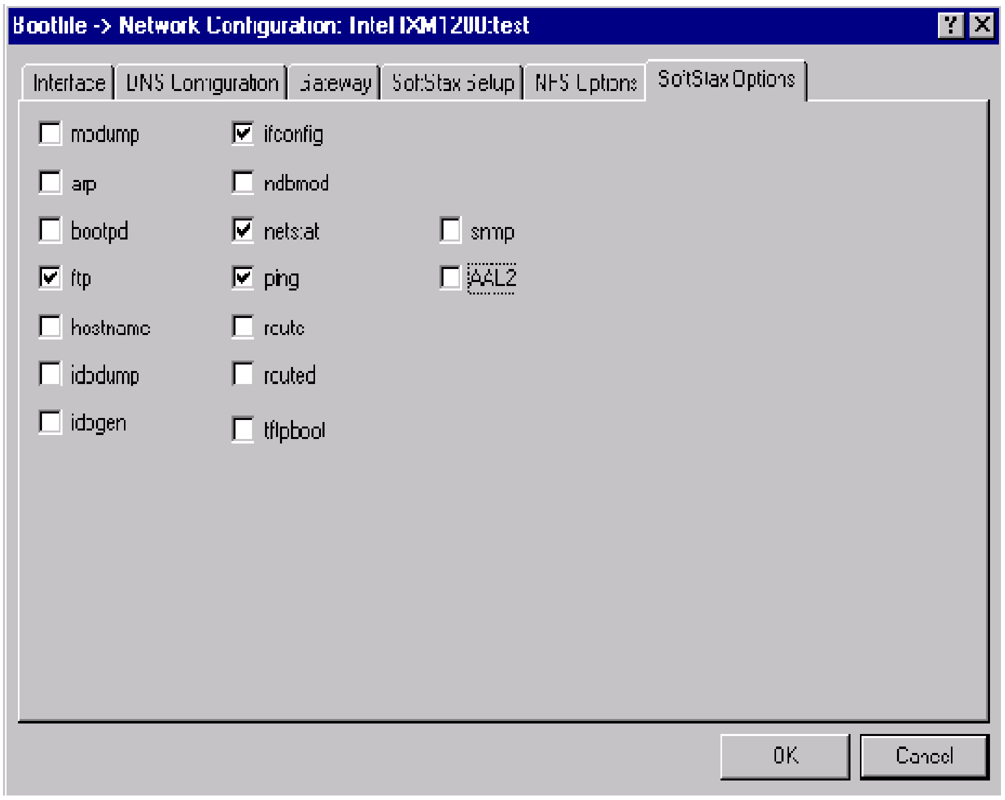
Adding AAL2 Support

To add AAL2 support to the bootfile image you created in the **Building a Bootfile for the IXM1200 Base Card** section, complete the following steps from the Configuration Wizard:

Step 1. From the Main Configuration window, select `Configure` -> `Bootfile` -> `Network Configuration`.

Step 2. From the **Network Configuration** dialog, select the `Softstax Options` tab. You should see the following screen:

Figure 1-9 SoftStax Options tab



Bootfile -> Network Configuration: Intel IXM1200:test

[ ] modump	[✓] ifconfig	
[ ] arp	[ ] ndbmod	
[ ] bootpd	[✓] netstat	[ ] snmp
[✓] ftp	[✓] ping	[ ] AAL2
[ ] hostname	[ ] route	
[ ] idbdump	[ ] routed	
[ ] idbgen	[ ] tftpbool	

Step 3. Select the AAL2 check box. This will add the following modules to your bootfile image:

```
<mwos>\OS9000\ARMV4\PORTS\IXP1200\CMDS\uenginit_atmaal2
<mwos>\OS9000\ARMV4\PORTS\IXP1200\CMDS\testaal2
<mwos>\OS9000\ARMV4\PORTS\IXP1200\CMDS\BOOTOBJS\SPF\spifixp
<mwos>\OS9000\ARMV4\PORTS\IXP1200\CMDS\BOOTOBJS\SPF\spifixp0
<mwos>\OS9000\ARMV4\PORTS\IXP1200\CMDS\BOOTOBJS\SPF\spixatm
<mwos>\OS9000\ARMV4\PORTS\IXP1200\CMDS\BOOTOBJS\SPF\spixatm0
```

Description of AAL2 Support Modules

Below is a description of the various OS-9 modules included in the bootfile image.

`uenginit`	enables loading and initializing microcode directly on the network processor’s microengines without using the Intel Developer’s Workbench.
`Testaal2`	is an OS-9 user-state application that emulates unidirectional transfer of various AAL2 CPS (Common Part Sublayer) packets between two AAL2 Service Access Points (SAP)). `Testaal2` allows you to verify data transfers over the AAL2 CPS SPF protocol stack. By default, `Testaal2` starts two separate, dedicated threads that simultaneously transmit and receive multiple variable length AAL2 CPS-Packets in loopback mode over Port 0 of IXF6012 ATM MAC.
`spifixp`	is the interface between the IXP1200 StrongARM core and microengines. This driver resides at the bottom of the SoftStax stack and pushes data to a protocol driver above it. Currently, the protocol drivers above `spifixp` include `spixeth` and `spixatm`.) In addition, the `spifixp` driver exists as a template for adding support for more protocols from the IXP1200 microengines.
`spifixp0`	is the descriptor of for the `spifixp` driver.

`spixatm`	uses the `spifixp` framework to support ATM packets transferred from the microcode to the OS-9 SoftStax system.
`Spixatm0`	is the descriptor for the `spixatm` driver.

Chapter 2: Board Specific Reference

This chapter contains porting information that is specific to the Intel IXM1200 Network Processor Base Card. It includes the following sections:

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

For More Information

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

Boot Options

The default boot options for the IXM1200 Network Processor Base Card 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 PCI Ethernet Card

The system can boot using the BootP protocol with an Ethernet card 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

Following is an example boot of the IXM1200 Network Processor Base Card using the `lr` boot option.

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

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 IXM1200 Network Processor Base Card.

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 rage shown below by the `irqixp1200` module.

For More Information

See the ***IXP1200 Network Processor Programer's Reference*** for further information on individual sources.

Table 2-1 OS-9 IRQ Assignment for the IXM1200 Network Processor Base Card

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
0x6	External Interrupt

Table 2-1 OS-9 IRQ Assignment for the IXM1200 Network Processor Base Card (continued)

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

Table 2-2 IXM1200 Network Processor Base Card Specific Functions

OS-9 IRQ #	IXM1200 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 IXM1200 Network Processor Base Card Specific Functions

OS-9 IRQ #	IXM1200 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 IXM1200 Network Processor Base Card Specific Functions

OS-9 IRQ #	IXM1200 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 IXM1200 Network Processor Base Card Specific Functions

OS-9 IRQ #	IXM1200 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)
0x7d	RTC (RTC block source)

Table 2-2 IXM1200 Network Processor Base Card Specific Functions

OS-9 IRQ #	IXM1200 Specific Function
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 SA1110 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.

GPIO Usage

The IXM1200 Network Processor Base Card has four GPIO pins. Each is connected to jumpers on the board. The 1200 GPIO unit is somewhat unique in that the GPIOs can be "owned" by either the StrongARM core or by the FBI unit. The owner of the GPIOs dictates their usability and function. The OS-9 boot code assigns all GPIOs to the StrongARM core and sets them up as inputs.

For More Information

See the ***IXP1200 Network Processor Programer's Reference*** for more information.

Table 2-3 GPIO Usage for the IXM1200 Network Processor Base Card

GPIO	Signal Name	Direct	Description
GPIO0	`CPU_GPIO_0`	Input	Value of switch SW2-1
GPIO1	`CPU_GPIO_1`	Input	Value of switch SW2-2
GPIO2	`CPU_GPIO_2`	Input	Value of switch SW2-3
GPIO3	`CPU_GPIO_3`	Input	Value of switch SW2-4

When the FBI unit owns particular GPIO pins, it indicates a certain MAC mode.

Table 2-4 MAC Modes

FBI owns GPIO #	MAC Mode
None 3-4	MAC mode, 64-bit bidirectional mode
GPIO [0] 1-2	MAC mode. 64-bit bidirectional mode
GPIO [1,2,3]	
GPIO [0,1,2,3]	32-Bit unidirectional mode

Port Specific Utilities

Utilities for the Spectacle Island Network Processor Evaluation Board 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 this board, 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
`-r`	program resident Flash (default)
`-p0`	program PCMCIA slot 0
`-p1`	program PCMCIA slot 1
`-ncis`	do not emit cis for PCMCIA Flash cards
`-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)
`-u`	leave Flash unlocked
`-i`	print out information on Flash
`-nv`	do not verify erase or write
`-q`	no progress indicator

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® IXP1200 Developer Workbench daemons for the Intel IXM1200 Network Processor Base Card are located in the following directory:

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

The following daemons allow the IXP1200 Developer Workbench to interact with the IXM1200 Network Processor Base Card:

`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 later versions of the IXP1200 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 IXP1200 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 IXM1200 Network Processor Base Card and on the Windows host system.

On the target system, your boot file (`os9kboot`) must include a properly configured Ethernet interface.

For More Information

See the **Building a Bootfile for the IXM1200 Base Card** section in **Chapter 1**.

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 IXP1200 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 IXM1200 Network Processor Base Card.

For More Information

See Appendix B: Running OS-9 and the IXP1200 Developer Workbench for more information.

`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. (Refer to the ***IXP1200 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-9000 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 Intel IXM1200 Network Processor Base Card. 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 Spectacle Island 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.
`pciconfig`	Initializes PCI bus devices.
`ioixp1200`	Provides low level serial access to the 1200's UART.

High-Level System Modules

The following OS-9 system modules are tailored specifically for the Intel IXM1200 Network Processor Base Card and peripherals. 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. See the **GPIO Usage** 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 IXP1200'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` IXP1200 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

See 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.

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, see the ***OS-9 Device Descriptor and Configuration Module Reference*** manual.

Appendix B: Running OS-9 and the IXP1200 Developer Workbench

This appendix provides a brief overview of using the IXP1200 Developer Workbench with an OS-9 system. It includes the following sections:

Overview

The OS-9 ROM image provides an interface with the Intel Core Software Library. This enables loading and debugging of microcode to the IXP1200 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 IXP1200 Network Processor development boards ship with the IXP1200 Developer Workbench provided by Intel.

For More Information

The IXP1200 Developer Workbench, which was shipped 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 described in **Chapter 1**.

System Configuration

Step 1. Install the IXP1200 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 Spectacle Island is booted up:

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

    Now trying to Override autobooters.

    Press the spacebar for a booter menu

    Now trying to Copy embedded OS-9000 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 &`. The following messages are displayed:

```
    +3
    $
    SPPRO100: Intel PCI EtherExpress Pro100 - PCI Device ID 0x1229
    SPPRO100: PCI device located @ BUS:DEV [0000:0007]
    SPPRO100: PCI I/O address 0x54000000
    SPPRO100: PCI DMA translation address 0x00000000
    SPPRO100: Transmit rings 0x40 Receive rings 0x40
    SPPRO100: Connection Type [INF_EXT] - Duplex mode [INF_AUTO_DUPLEX]
    SPPRO100: Processor cache line flushing enabled
    SPPRO100: Board assembly =  721383-10
    SPPRO100: Physical connectors present:  RJ45
    SPPRO100: Primary interface chip  i82555
    SPPRO100: MII - PHY # 0x01
    SPPRO100: Receiver lock-up workaround not required. [0x0203 0x0200 0x07B3]
   SPPRO100: Memory allocated @ 0xC0FE68A0 - Size 0x00019760 - Color 0x00000002
```

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

Step 5. Start the first IXP1200 Developer Workbench daemon by typing the following command at the OS-9 prompt: `ixp_engine &`.

Step 6. Start the second IXP1200 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 IXM1200 Network Processor Base Card.

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 IXP1200 Developer Workbench opens in the foreground.

Step 4. From the IXP1200 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 IXP1200 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.

IXP1200 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.

IXP1200 StrongARM Core Software Libraries

Functions for the Intel IXP1200 StrongARM core software libraries are located in the following directory:

```
MWOS/OS9000/ARMV4/PORTS/IXP1200/LIB/netapp.l
```

For More Information

For more information regarding the Intel IXP1200 libraries, refer to the ***IXP1200 Software Reference Manual*** and library source (`IXP1200\SA1_CoreLibs`) distributed with the IXP1200 Developer's Workbench.