Home

OS-9® for IXDP425 Board Guide

Version 4.7

RadiSys
THE POWER OF WE

www.radisys.com

Revision A • July 2006

Copyright and publication information

This manual reflects version 4.7 of Microware OS-9.
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.

Contents

: Contents

Chapter 1: Installing and Configuring OS-9®

Development Environment Overview **6**
Requirements and Compatibility **7**
Host Hardware Requirements (PC Compatible) 7
Host Software Requirements (PC Compatible) 7
Target Hardware Requirements 7
Target Hardware Setup **8**
RedBoot Flash part 8
Connecting the Target to the Host **9**
Building the ROM Image **11**
Coreboot 11
Bootfile 11
Starting the Configuration Wizard 12
Creating the ROM Image 13
Creating the Coreboot Image 14
Creating the Bootfile Image 14
Building a ROM Image 14
Bootstrapping OS-9 ..... **17**
Burning the Flash Part **18**
Programming a ROM Image with RedBoot 19
Programming a ROM Image with the pflash Utility 20

Chapter 2: Board-Specific Reference

Boot Options **24**
Booting from Flash 24
Booting over a Serial Port via kermit 24
Restart Booter 25
Break Booter 25
Sample Boot Session and Messages 25
The Fastboot Enhancement **26**
Overview 26
Implementation Overview 27

B_QUICKVAL ... 27
B_OKRAM ... 27
B_OKROM ... 27
B_1STINIT ... 28
B_NOIRQMASK ... 28
B_NOPARITY ... 28
Implementation Details ... 29
Compile-time Configuration ... 29
Runtime Configuration ... 29
OS-9 Vector Mappings ... 30
Fast Interrupt Vector (0x7) ... 32
Port Specific Utilities ... 33
dmppci ... 34
pciv ... 35
pflash ... 36
setpci ... 37

Appendix A: Board-Specific Modules

Low-Level System Modules ... 40
High-Level System Modules ... 41
CPU Support Modules ... 41
System Configuration Module ... 41
Interrupt Controller Support ... 42
Ticker ... 42
Generic I/O Support Modules (File Managers) ... 42
Pipe Descriptor ... 43
RAM Disk Support ... 43
RAM Descriptors ... 43
Serial and Console Devices ... 43
Descriptors for use with scixc1100 ... 44
Descriptors for use with scllio ... 44
SPF Device Support ... 45
Support for NPE ports ... 45
spethix Descriptors ... 45
Network Configuration Modules ... 45
Port Specific Utilities ... 45
Common System Modules List ... 46

1 Installing and Configuring OS-9®

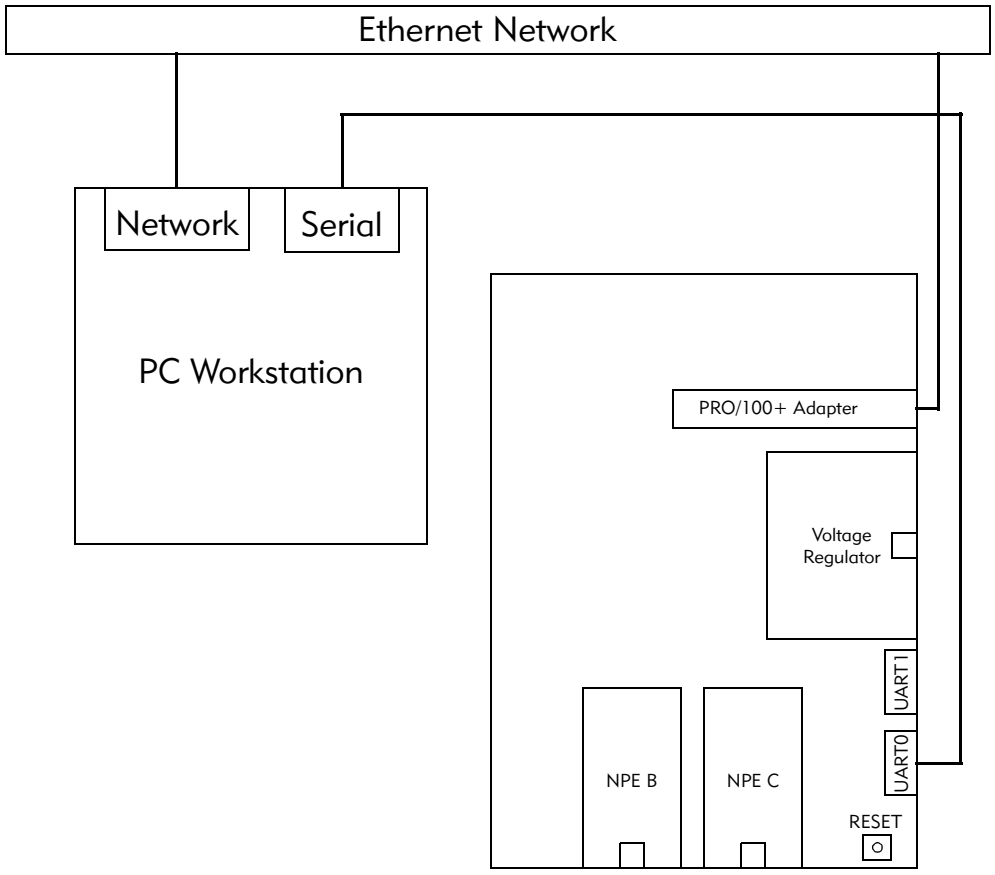
This chapter describes how to install and configure OS-9 on the Intel IXDP425 development platform. It includes the following sections:

- Development Environment Overview
- Requirements and Compatibility
- Target Hardware Setup
- Connecting the Target to the Host
- Building the ROM Image
- Bootstrapping OS-9
- Burning the Flash Part

Development Environment Overview

Figure 1-1 shows a typical development environment for the Intel IXDP425. The components shown are the minimum required to develop software with OS-9 and the IXDP425.

Figure 1-1. IXDP425 Development Environment



Requirements and Compatibility

Before you begin, install the *Microware OS-9 for XScale* 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:

- Microware OS-9 for XScale
- Windows 95, Windows 98, Windows NT 4.0, Windows 2000, Windows ME, or Windows XP.
- terminal emulation program

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

- BOOTP server. The BOOTP server is required to assign an IP address as RedBoot brings the board up. In addition, if Ethernet booting of OS-9 is required the BOOTP server will also be used.
- TFTP server. The TFTP server is required to serve boots to the IXDP425 via RedBoot bootstrap sequence. Again, if Ethernet booting of OS-9 is required the TFTP server will also be used.

Target Hardware Requirements

Your reference board requires the following Intel supplied equipment:

- Intel IXDP425 board

- power supply
- serial cable
- Intel Ethernet Pro 100+ PCI card
- Ethernet cable
- Access to an Ethernet network

Target Hardware Setup

RedBoot Flash part

Install the Intel-supplied RedBoot Flash part into the IXDP425. Follow the instructions provided by Intel. The RedBoot boot monitor is used to bootstrap OS-9 onto the board.

Be very careful when handling the Flash chips as the pins can be easily bent.

Connecting the Target to the Host

To connect the target to your host machine, complete the following steps:

Step 1. Ensure that the power switch on the power supply is in the OFF position. Connect the power cord to the power supply and connect the power supply to the IXDP425 board.

Step 2. Install the Intel Ethernet Pro 100+ PCI card in the bottom PCI slot marked PCI SLOT 3.

Step 3. Plug one end of your Ethernet cable into the RJ-45 connector on the PCI card and the other end into your Ethernet network.

Step 4. Connect the male end of the provided serial cable into the 9-bin serial port marked UART0. Connect the female end into your PC workstation.

Step 5. On the Windows desktop, click the `Start` button and select `Programs -> Accessories -> Communications -> Hyperterminal`. Select the `Hyperterminal` icon.

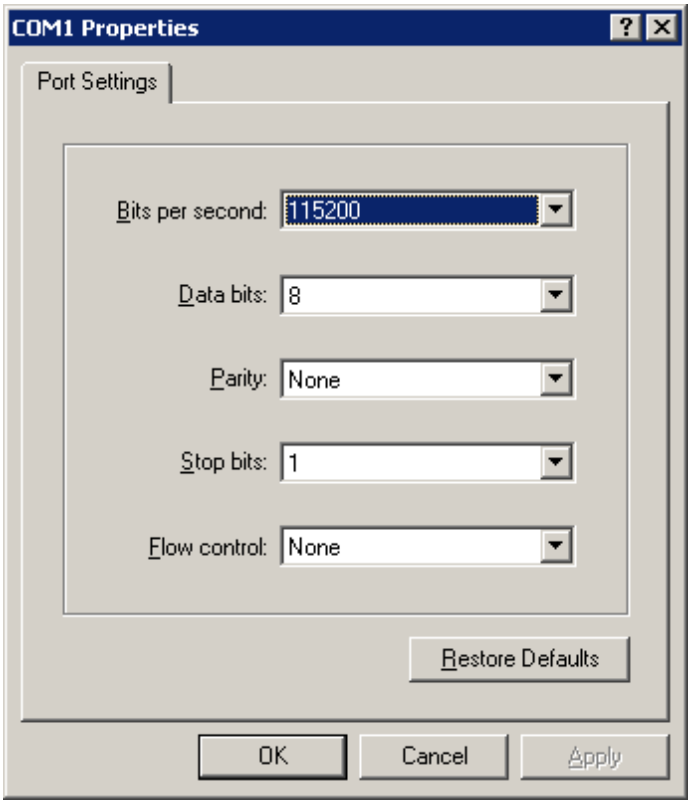
Step 6. Enter a name for your Hyperterminal session and select an icon for the new session. Click `OK`. A new icon with the name of your session displays. The settings you choose for this session can be saved for future use.

Step 7. In the **Connect To** dialog, go to the **Connect Using** box and select the communications port with which you plan to connect to the reference board. The port you select must be the same port in which you inserted the cable to your host machine. Click `OK`.

Step 8. In the **Properties** box on the **Port Settings** tab (shown in Figure 1-2), enter the following settings, then click `OK` to close the dialog.

```
Bits per second = 115200
Data Bits = 8
Parity = None
Stop bits = 1
Flow control = None
```

Figure 1-2. Port Settings



Step 9. Go to the **Hyperterminal** menu and select `Call -> Call` 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 displays the word *Connected*.

Step 10. Apply power to the board. The board should fail to find a BOOTP server. Output similar to the following displays:

```
FLASH configuration checksum error or invalid key
Ethernet eth0: MAC address 00:07:e9:0d:95:be
Can't get BOOTP info for device!

RedBoot(tm) bootstrap and debug environment [ROM]
Non-certified release, version 2.00 - built 17:25:55, Dec 13 2002

Platform: Intel(R) IXDP425
Copyright (C) 2000, 2001, 2002, Red Hat, Inc.

RAM: 0x10100000-0x20000000, 0x101156a8-0x1ffdd000 available
FLASH: 0x50000000 - 0x51000000, 128 blocks of 0x00020000 bytes each.
RedBoot>
```

Step 11. Record the MAC address (00:07:e9:0d:95:be in this example) for later use.

Step 12. Create an entry in the bootptab file for your BOOTP server that will allow the BOOTP server to respond to the recorded MAC address and provide it with an internet address. The boot filename is irrelevant—it is not used by RedBoot.

Step 13. Press the reset button to reboot the IXDP425 board. This time it should find the BOOTP server and be served the internet address you specified in the bootptab file.

Your target is now ready to download an OS-9 ROM image from your TFTP server.

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:

- coreboot, the low-level image.
- bootfile, the high-level image.

Coreboot

The coreboot image generally initializes hardware devices and locates the high-level (or bootfile) image as specified by its configuration. Depending on hardware capabilities, the bootfile image may be found on a Flash part, a hard disk, or on an Ethernet network. It also builds basic structures based on the image it finds and passes control to the kernel to bring up the OS-9 system.

Bootfile

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

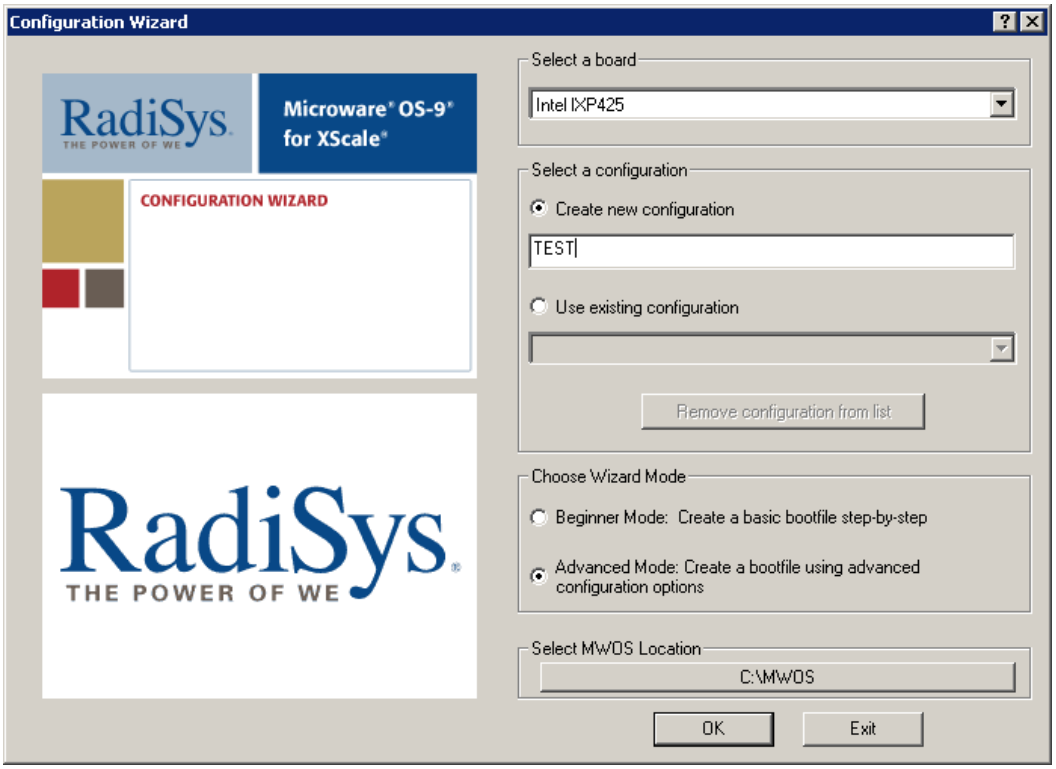
Microware provides a Configuration Wizard to create a coreboot image, a bootfile image, or an entire OS-9 ROM Image. The Wizard can also modify an existing image. The Configuration Wizard automatically installs on your host PC during the installation process.

Starting the Configuration Wizard

The Configuration Wizard builds the coreboot, bootfile, or ROM image. To start the Wizard, perform the following steps:

Step 1. From the Windows desktop, select `Start -> Programs -> RadiSys -> Microware OS-9 for XScale -> Microware Configuration Wizard`.

Figure 1-3. Configuration Wizard Opening Screen

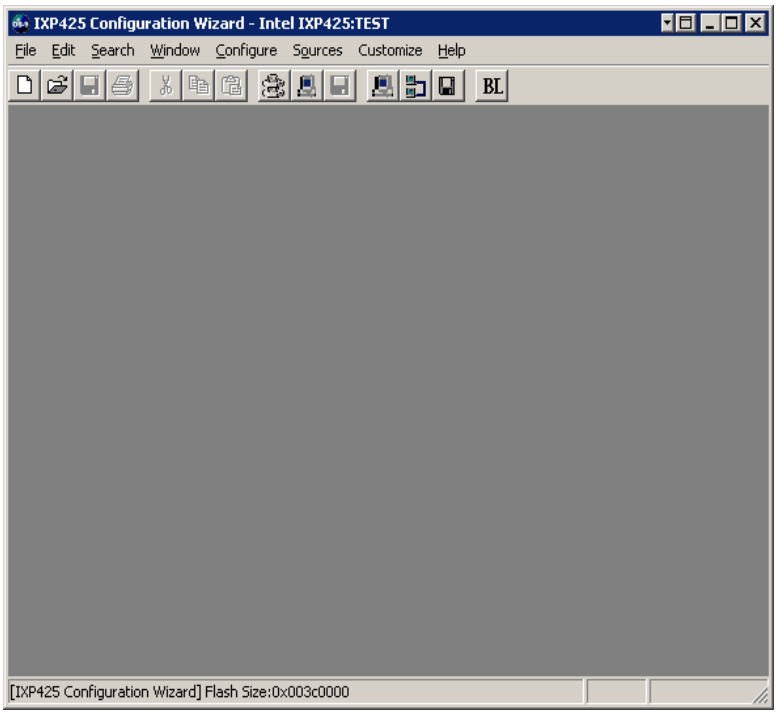


Step 2. Select your target board, Intel IXP425, from the **Select a board** pull-down menu.

Step 3. Select the `Create new configuration` radio button from the **Select a configuration** menu and enter a name for your boot configuration in the supplied text box. This names your new configuration, which you can access later by selecting the **Use existing configuration** pull-down menu.

Step 4. Select the `Advanced Mode` radio button from the **Choose Wizard Mode** group and click `OK`. The Wizard's main window displays. This is the window from which you build your image. An example is shown in the following figure.

Figure 1-4. Configuration Wizard Main Window



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. You use the Configuration Wizard to choose the contents of your OS-9 implementation. You can also create individual coreboot and bootfile images, or combine them into a single file called the ROM image.

Creating the Coreboot Image

The default Intel IXP425 coreboot configuration is valid and does not require modification to create a working system.

Creating the Bootfile Image

The default settings in the Configuration Wizard are preset for optimum performance for the Intel IXP425 XScale board. The only modifications required are the network settings. The network settings information must be obtained from your network administrator.

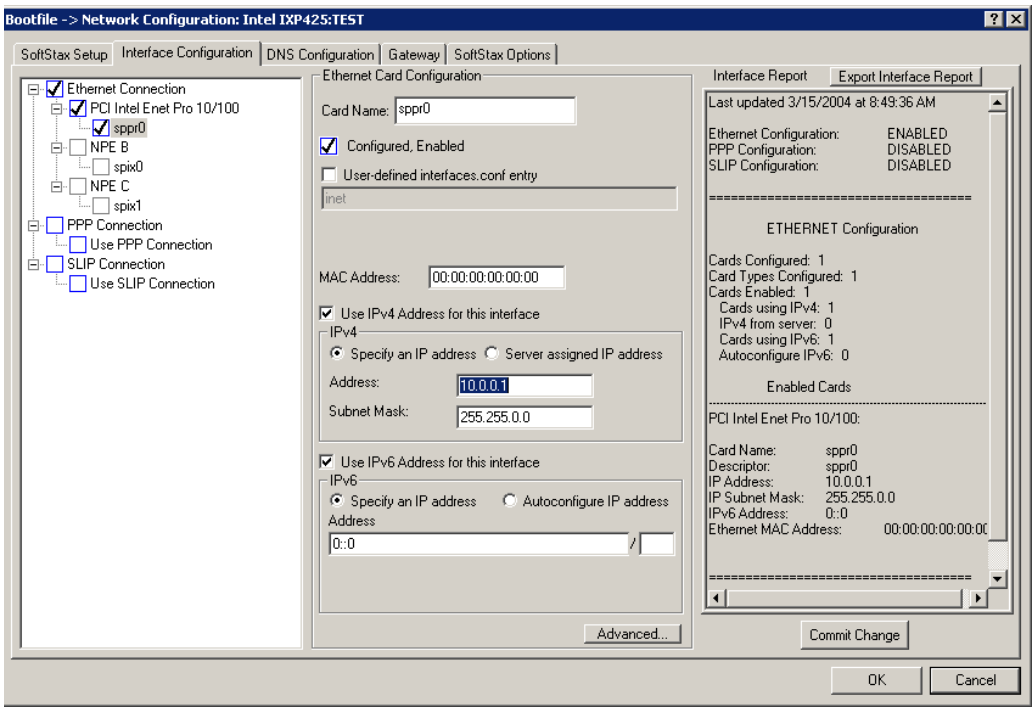
Building a ROM Image

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

Step 1. To use the target board across a network, you must configure the Ethernet settings within the Configuration Wizard. To do this, select `Configure -> Bootfile -> Network Configuration` from the Wizard's main menu.

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

Figure 1-5. Network Configuration Dialog

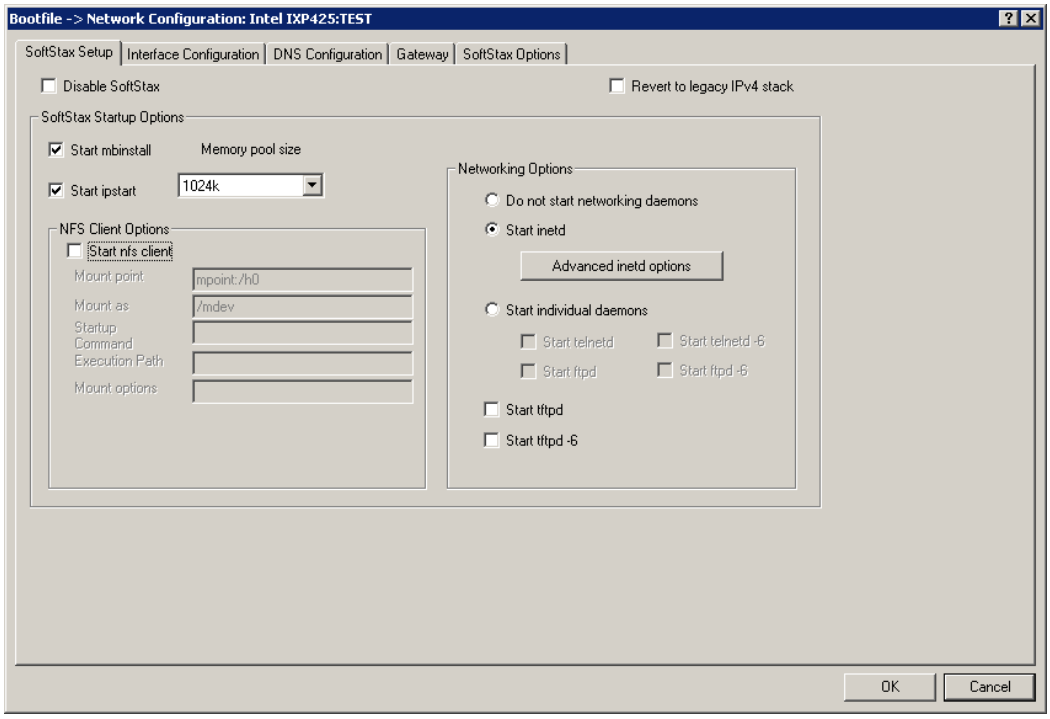


To learn more about IPv4 and IPv6 functionalities, see the *Using LAN Communications* manual included with this product CD.

If you do not know the network values for your board, contact your system administrator.

Step 3. Select the **SoftStax® Setup** tab. The **SoftStax Setup** tab displays as shown in the next figure.

Figure 1-6. SoftStax Setup Tab



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

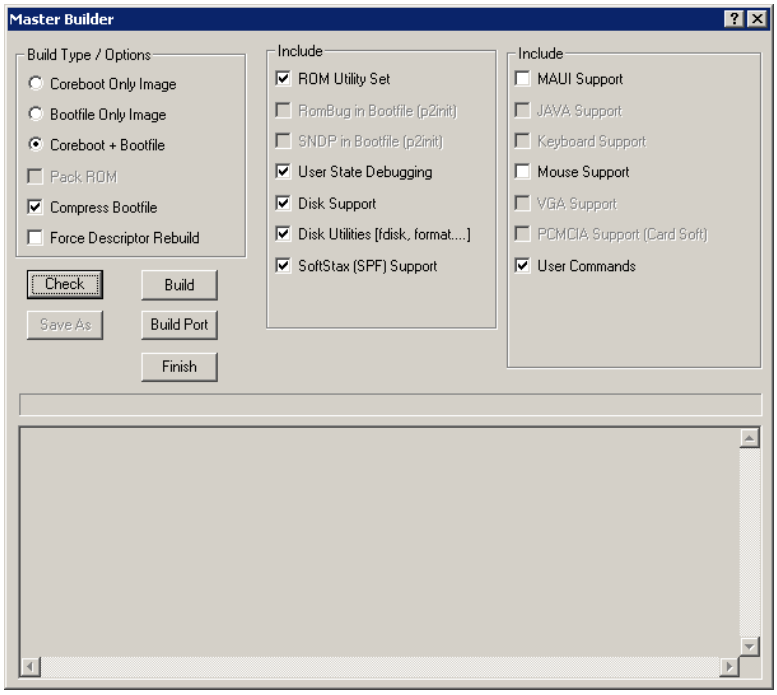
Step 5. Leave the other **Network Configuration** options at the default settings.

Change other network configuration options in this dialog, such as DNS and gateway, according to your specific requirements and network.

Step 6. Click `OK` in the **Network Configuration** dialog to close it and return to the **Configuration Wizard** main window.

Step 7. Select `Configure -> Build Image`. The **Master Builder** dialog window displays as shown in the next figure.

Figure 1-7. Master Builder Dialog



Step 8. Configure your Master Builder options as shown in the figure above.

Step 9. Click `Build`. This builds a ROM image that the RedBoot boot monitor can load. The image, `rom`, is stored in the following directory:

```
\MWOS\OS9000\ARMV5\PORTS\IXP425\BOOTS\INSTALL\PORTBOOT
```

Use the Save As... button to save the `rom` file to the correct location for your TFTP server.

Bootstrapping OS-9

The target is now configured, the `rom` file is saved, and you are ready to boot OS-9 on your IXDP425.

Step 1. Turn the power supply on or press the reset button on the IXDP425.

Step 2. Wait for the `RedBoot>` prompt and then type `load -r -b 0x10200000 rom`.

Step 3. Wait for the download to finish and then type `go 0x10200000`.

OS-9 boots and networking is available. The following is an example of the output:

```
FLASH configuration checksum error or invalid key
Ethernet eth0: MAC address 00:07:e9:0d:95:be
IP: 208.252.117.7, Default server: 208.252.116.54

RedBoot(tm) bootstrap and debug environment [ROM]
Non-certified release, version 2.00 - built 17:25:55, Dec 13 2002

Platform: Intel(R) IXDP425
Copyright (C) 2000, 2001, 2002, Red Hat, Inc.

RAM: 0x10100000-0x20000000, 0x101156a8-0x1ffdd000 available
FLASH: 0x50000000 - 0x51000000, 128 blocks of 0x00020000 bytes each.
RedBoot> load -r -b 0x10200000 rom
Raw file loaded 0x10200000-0x103d164c
RedBoot> go 0x10200000


OS-9 Bootstrap for the ARM (Edition 68)

PCI device initialization - Completed
Now trying to Override autobooters.

Press the spacebar for a booter menu

Now trying to Boot embedded OS-9 in-place.
Compressed bootfile found at $10300000
A valid OS-9 bootfile was found.
+3
+5
$
```

Burning the Flash Part

You may want to burn a ROM file into the Flash part. This is done by creating a Flash image with RedBoot's `fis create` command. The ROM image can then be loaded without the need for a BOOTP server nor a TFTP server.

There are no special requirements when creating boots that will reside in Flash.

There are two ways to burn the ROM image into the Flash part: with RedBoot and with OS-9's `pflash` utility. Using the RedBoot method at least one time results in an easier to maintain system since it allocates a section of the Flash for OS-9's use. Each of these methods will be discussed in the following two sections.

Programming a ROM Image with RedBoot

The following steps assume that your BOOTP and TFTP server are functional and you have already been able to boot OS-9 on your IXDP425.

Step 1. Apply power to your system or press the reset button if the system is already on. Wait for the `RedBoot>` prompt.

Step 2. Issue the `fis init` command to initialize the flash image system (FIS) directory structure in the Flash part.

```
RedBoot> fis init
About to initialize [format] FLASH image system - are you sure (y/n)? y
*** Initialize FLASH Image System
    Warning: device contents not erased, some blocks may not be usable
... Unlock from 0x50fe0000-0x51000000: .
... Erase from 0x50fe0000-0x51000000: .
... Program from 0x1ffdf000-0x1ffdf300 at 0x50fe0000: .
... Lock from 0x50fe0000-0x51000000: .
RedBoot>
```

Step 3. Load the OS-9 ROM image into RAM.

```
RedBoot> load -r -b 0x10200000 rom
Raw file loaded 0x10200000-0x103d164c
```

Step 4. Use the `fis create` command to create a new image in the FIS directory structure. The new image is created at the midpoint of the Flash part (0x50800000).

```
RedBoot> fis create -b 0x10200000 -l 0x3c0000 -f 0x50800000 -r 0x10200000
os9
... Erase from 0x50800000-0x50bc0000: ..............................
... Program from 0x10200000-0x105c0000 at 0x50800000: ..................
... Unlock from 0x50fe0000-0x51000000: .
... Erase from 0x50fe0000-0x51000000: .
... Program from 0x1ffdf000-0x1ffff000 at 0x50fe0000: .
... Lock from 0x50fe0000-0x51000000: .
```

Step 5. Turn off the power to the board for a short time.

Step 6. Turn the power back on and wait for the `RedBoot>` prompt. At the prompt use the `fis load` command to load the OS-9 ROM image into RAM.

```
RedBoot> fis load os9
```

This `fis load` command accomplishes exactly the same thing as the TFTP `load` command used previously, except that it doesn't require a BOOTP nor a TFTP server.

Programming a ROM Image with the pflash Utility

`pflash` is an OS-9 utility that transfers an image into Flash. The following steps detail how to create a new coreboot image and burn the image into Flash using this utility.

Step 1. If no longer running the Configuration Wizard (from the previous steps) on your Windows desktop, select `Start -> Programs -> RadiSys -> Microware OS-9 for XScale -> Microware Configuration Wizard`. The Configuration Wizard opening screen displays. Click on the `Use existing configuration` radio button in the **Select a configuration** group. Ensure that your previous configuration appears in the drop-down menu and that the `Advanced Mode` radio button is selected in the **Choose Wizard Mode** group. Click `OK`.

Step 2. Select `Configure -> Build Image..` to display the **Master Builder** screen.

Step 3. Ensure `Coreboot + Bootfile` is selected and click `Build`.

Step 4. Once the build is complete, click `Save As` to save the ROM image to a directory of your choosing.

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

```
\MWOS\OS9000\ARMV5\PORTS\IXP425\BOOTS\INSTALL\PORTBOOT
```

Step 5. Start a DOS shell on the host system.

Step 6. Navigate to the directory in which the OS-9 ROM image, `rom`, is located.

Step 7. On the target, initialize the large RAM disk (`/r1`) so it can hold the ROM image by entering the following command in the Hyperterminal window:

```
$ iniz /r1
```

Step 8. On your Windows host, use FTP to transfer the new image to the target system. At the prompt enter the following command:

```
ftp <IP address or host name of target>
```

Step 9. Log in with the username `super` and the password `user`.

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

```
ftp> cd /r1
```

This specifies that the `/r1` device 's root is the current directory.

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

```
ftp> bin
```

This designates binary format.

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

```
ftp> put rom
```

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

Step 13. On the target, enter the following command:

```
$ pflash -s=0x50800000 -f=/r1/rom
```

The file is programmed into the target system's Flash memory. It can be loaded with RedBoot's `fis load` command previously described.

2 Board-Specific Reference

This chapter contains porting information specific to the Intel IXDP425 Embedded XScale board. It includes the following sections:

- Boot Options
- The Fastboot Enhancement
- OS-9 Vector Mappings
- Port Specific Utilities

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

Boot Options

Default boot options for the Intel IXDP425 are listed below. The boot options can be selected by pressing the space bar during system boot when the following message appears on the serial console:

```
Press the spacebar for a booter menu
```

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

```
MWOS\OS9000\ARMV5\PORTS\IXP425\ROM
```

Booters can be configured to be either of these:

- Auto booters, which automatically attempt to boot in the same order as listed in the auto booter array.
- Menu booters, from the defined menu booter array, which are chosen interactively from the console command line after the boot menu displays.

Booting from Flash

When the `rom_cnfg.h` file has a defined ROM search list, the options `bo` and `lr` appear in the boot menu. If no ROM 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\ARMV5\PORTS\IXP425\ROM\ROMCORE
```

`bo`	ROM boot—the system runs "in-place", which on the IXDP425 is actually RAM.
`lr`	load to RAM—the system copies the ROM bootfile image into RAM and 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 duration of this transfer depends of the bootfile's size, but it usually takes at least three minutes to complete. Dots on the console indicate download progress.

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

Restart Booter

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

`q`	quit: Quit and try 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 resets.

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

Sample Boot Session and Messages

Below is an Intel IXDP425 example boot using the `bo` boot option.

```
OS-9 Bootstrap for the ARM (Edition 68)

PCI device initialization - Completed
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>
Enter system debugger -------------- <break>
Restart the System ----------------- <q>

Select a boot method from the above menu: bo

Compressed bootfile found at $10300000
A valid OS-9 bootfile was found.
```

```
+3
+5
$ mfree
Current total free RAM: 183540.00 K-bytes
```

The Fastboot Enhancement

Fastboot enhancements to OS-9 provide faster system bootstrap performance to embedded systems. Normal OS-9 bootstrap performance 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, bootstrap speed achieves a significant increase.

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 to omit the associated bootstrap functionality.

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 are implemented as a set of bit fields. An entire 32-bit field is 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, six bit flags are 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 module headers of ROM modules are 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 drastically decreases bootstrap time. Data corruption rarely occurs in ROM. Therefore, omitting CRC checking is usually a safe option.

B_OKRAM

The `B_OKRAM` bit informs both low- and high-level systems that they should accept their respective RAM definitions without verification. Normally, the system probes memory during bootstrap based on 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 functional, the defined RAM list does not need validating. Bootstrap occurs faster 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 by 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 kernel cold-start mask interrupts for the duration of system startup. However, some systems with a well-defined interrupt system (i.e. completely calmed by the sysinit hardware initialization code) and also 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.

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 is not 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 require either no parity initialization at all or systems that require it only for "power-on" reset conditions. Systems that require parity initialization only 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 predefined macro (`BOOT_CONFIG`), used to set the initial bit-field values of bootstrap flags. You can redefine the macro for recompilation to create a new bootstrap configuration. The new overriding 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 configuration details of the low-level system. Below is an example of how to redefine the system's bootstrap configuration 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 RAM and ROM definitions without verification and 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 the user-defined bootstrap procedure to use. An example P2 module is shown below. If the override is performed in the `sysinit2()` function, the effect is not realized until after the low-level system memory searches are 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 IXDP425 Embedded XScale board.

The ARM standard defines exceptions 0x0-0x7. The OS-9 system maps these one-to-one. External interrupts from vector 0x6 expand to the virtual vector range shown below by the `irqixc1100` module.

Table 2-1. OS-9 IRQ Assignment for the Intel IXDP425

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
0x7	Fast Interrupt
0x8	Alignment Error Form of Data abort

Table 2-2. Intel IXDP425 Specific IRQ Assignments

OS-9 IRQ #	IXDP425 Specific IRQ
0x41	Ethernet NPE A
0x42	Ethernet NPE B
0x43	Queue Manager Queues 1 - 32
0x44	Queue Manager Queues 33 - 64
0x45	General Purpose Timer 0
0x46	GPIO[0]
0x47	GPIO[1]
0x48	PCI Interrupt
0x49	PCI DMA Channel 1
0x4a	PCI DMA Channel 2
0x4b	General Purpose Timer 1
0x4c	USB
0x4d	Console UART (UART1)
0x4e	Timestamp Timer
0x4f	High-speed UART (UART0)
0x50	Watchdog Timer
0x51	Performance Monitoring Unit Counter Rollover
0x52	XScale PMU Counter Rollover
0x53	GPIO[2]
0x54	GPIO[3]
0x55	GPIO[4]
0x56	GPIO[5]
0x57	GPIO[6]
0x58	GPIO[7]
0x59	GPIO[8]
0x5a	GPIO[9]
0x5b	GPIO[10]
0x5c	GPIO[11]
0x5d	GPIO[12]
0x5e	Software Interrupt 0
0x5f	Software Interrupt 1

Fast Interrupt Vector (0x7)

The ARM5-defined fast interrupt (FIQ) mapped to vector 0x7 is handled differently by OS-9 interrupt code and cannot 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 used. This requirement means that a FIQ handler's entry and exit points must be in assembly, as the C compiler makes assumptions about context. In addition, system calls are not possible unless a full C ABI context save is first performed. The OS-9 IRQ code for the XScale assigns all interrupts as normal external interrupts. It is up to the user to redefine a source as an FIQ to make use of this feature.

Port Specific Utilities

Utilities for the Intel IXDP425 are located in the following directory:

`MWOS/OS9000/ARMV5/PORTS/IXP425/CMDS`

The following port specific utilities are included:

`dmppci`	Shows 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>}
```

Description

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

pciv
PCI Configuration Space View

Syntax

```
pciv [options]
```

Options

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

Description

The `pciv` utility allows visual indication of the status of the PCIbus. This should be executed only on a PCI host device.

pflash
Program Strata Flash

Syntax

```
pflash [options]
```

Options

`-f[=]filename`	Input filename (required parameter).
`-eu`	Erase used space only (default).
`-ew`	Erase whole flash.
`-ne`	Do not erase flash.
`-i`	Print information about flash.
`-nv`	Do not verify erase or write operations.
`-q`	Do not display progress indicator.
`-b[=]addr`	Specify base address of flash (hex). The default is 0x50000000.
`-s[=]addr`	Specify write/erase address for file (hex). The default start address is 0x50000000.
`-u`	Leave Flash part unlocked.
`-np`	Disable protection. Reprograms addresses in the range 0x5000000 to 0x50040000 or 0x50fc0000 to 0x51000000. Note: These protected areas are used for the RedBoot code and FIS directory structure. Reprogramming them may make your board unbootable.

Description

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

setpci
Set PCI Value

Syntax

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

Description

The `setpci` utility sets PCI configuration information not normally available by other means, other than programming with the PCI library. The `setpci` utility can also 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 (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 executes in read mode.

A Board-Specific Modules

This chapter describes 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 IXDP425. The functionality of many modules can be altered through changes to the configuration data module (`cnfgdata`). These modules are located in the following directory:

`MWOS/OS9000/ARMV5/PORTS/IXP425/CMDS/BOOTOBJS/ROM`

`cnfgdata`	Contains the low-level configuration data.
`cnfgfunc`	Provides access services to `cnfgdata` data.
`commcnfg`	Initializes the communication port defined in `cnfgdata`.
`conscnfg`	Initializes the console port defined in `cnfgdata`.
`dumppciccs`	Displays the contents of the PCI Command and Configuration Space.
`initext`	Performs basic initialization of the PCI sub-system.
`io16550`	Provides low-level serial services via the on-board UARTs.
`llpro100`	Drives the Intel Ethernet Pro 10/100 PCI board. This driver is used for low-level debugging and Ethernet booting.
`pciwalk`	Walks the PCI bus, plus partitions PCI memory and I/O space to found devices. When the board operates in option (non-host) mode, this module performs no function.
`portmenu`	Initializes booters defined in `cnfgdata`.
`romcore`	Provides board-specific initialization code.
`tmrixc1100`	Uses general purpose timer 0 to provide timer services for the low-level system.
`usedebug`	Initializes low-level debug interface to RomBug, SNDP, or none.

High-Level System Modules

The following OS-9 system modules are tailored specifically for the Intel IXDP425. Unless otherwise specified, each module is located in a file of the same name in the following directory:

`MWOS/OS9000/ARMV5/PORTS/IXP425/CMDS/BOOTOBJS`

CPU Support Modules

These files are located in the following directory:

`MWOS/OS9000/ARMV5/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 `cachexscale` file.
`fpu`	Provides software emulation for floating point instructions.
`ssm`	System Security Module—provides support for the CPU's MMU (Memory Management Unit).
`vectors`	Provides interrupt service entry and exit code. The `vectors` module is found in the file `vectxscale`.

System Configuration Module

The system configuration modules are located in the following directory:

`MWOS/OS9000/ARMV5/PORTS/IXP425/CMDS/BOOTOBJS/INITS`

`dd`	High-level initialization module that specifies `/dd` as the initial disk device.
`nodisk`	High-level initialization module that specifies no initial disk device. This `init` module is used in diskless systems.
`configurer`	Init module generated by the Configuration 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. For more information, see the Port Specific Utilities section.

`irqixc1100`	System extension module that provides interrupt dispatching support for the IXC1100's interrupt controller (vector range 0x40 - 0x5f).

Ticker

`tkixc1100`	Driver that provides the system ticker based on the IXC1100's general purpose timer 1.
`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/ARMV5/CMDS/BOOTOBJS`

`ioman`	Generic I/O support for all I/O device types.
`scf`	Character device management functions.
`rbf`	Generic block device management functions for the OS-9 format.
`pcf`	Block device management functions for MS-DOS FAT format.
`spf`	Generic protocol device management function support.
`pipeman`	Memory FIFO buffer for communication.

Pipe Descriptor

The pipe descriptor is located in the following directory:

`MWOS/OS9000/ARMV5/PORTS/IXP425/CMDS/BOOTOBJS/DESC`

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

RAM Disk Support

The RAMdisk device driver is located in the following directory:

`MWOS/OS9000/ARMV5/CMDS/BOOTOBJS`

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

RAM Descriptors

The RAMdisk descriptors are located in the following directory:

`MWOS/OS9000/ARMV5/PORTS/IXP425/CMDS/BOOTOBJS/DESC/RAM`

`r0`	RBF descriptor that provides access to a 512K RAM disk.
`r0.dd`	RBF descriptor that provides access to a 512K RAM disk—with module name `dd` (for use as the default device).
`r1`	RBF descriptor that provides access to an 8MB RAM disk for storing images to program into the Flash.

Serial and Console Devices

`scixc1100`	SCF driver that provides serial support the IXC1100's internal UARTs.

Descriptors for use with scixc1100

`term1/t1` — Descriptor modules for use with `scixc1100`

UART0 (Fast port)

Default Baud Rate:	115200
Default Parity:	None
Default Data Bits:	8
Default Stop Bits:	1
Default Handshake:	XON/XOFF

`term2/t2` — Descriptor modules for use with `scixc1100`

UART1 (Console port)

Default Baud Rate:	115200
Default Parity:	None
Default Data Bits:	8
Default Stop Bits:	1
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\ARMV5\PORTS\IXP425\CMDS\BOOTOBJS\DESC\SCLLIO`

`vcons/term` — Descriptor modules for use with `scllio` in conjunction with a low-level serial driver. Port configuration and setup follows that which is configured in `cnfgdata` for the console port. `scllio` can communicate with a true low-level serial device driver like `io16550`, 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*.

SPF Device Support

Support for NPE ports

The Ethernet support module is located in the following directory:

`MWOS/OS9000/ARMV5/PORTS/IXP425/CMDS/BOOTOBJS/SPF`

`spethix`	SPF driver to support ethernet via NPE B or NPE C.

spethix Descriptors

These descriptor files are located in the following directory:

`MWOS/OS9000/ARMV5/PORTS/IXP425/CMDS/BOOTOBJS/SPF`

`spix0`	SPF descriptor module for use with the NPE B Ethernet connector.
`spix1`	SPF descriptor module for use with NPE C Ethernet connector.

Network Configuration Modules

These files are located in the following directory:

`MWOS/OS9000/ARMV5/PORTS/IXP425/CMDS/BOOTOBJS/SPF`

`inetdb`	Internet database information.
`inetdb2`	Target-specific database information.
`rpcdb`	RPC database information.

Port Specific Utilities

The following IXDP425-specific programs are provided. For more information about their functions and syntax, enter the `-?` command-line option. They are located in the following directory:

`MWOS/OS9000/ARMV5/PORTS/IXP425/CMDS`

`dmppci`	Allows a specific PCI device's configuration area to display.

`pciv`	Displays configuration information about all available PCI devices.
`pflash`	Programs the on-board Intel StrataFlash device.
`setpci`	Allows changes to the PCI configuration of devices.

Common System Modules List

The following low-level system modules provide generic services for OS-9 Modular ROM. They are located in the following directory:

`MWOS/OS9000/ARMV5/CMDS/BOOTOBJS/ROM`

`bootsys`	Provides booter registration services.
`console`	Provides console services.
`dbgentry`	Inits debugger entry point for system use.
`dbgserv`	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 a 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 the 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 a complete list of OS-9 modules common to all boards,see the *OS-9 Device Descriptor and Configuration Module Reference* manual.