



[Home](#)

OS-9[®] for EXS-6424 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.

July 2006
Copyright ©2006 by RadiSys Corporation
All rights reserved.

EPC and RadiSys are registered trademarks of RadiSys Corporation. ASM, Brahma, DAI, DAQ, MultiPro, SAIB, Spirit, and ValuePro are trademarks of RadiSys Corporation.

DAVID, MAUI, OS-9, OS-9000, and SoftStax are registered trademarks of RadiSys Corporation. FasTrak, Hawk, and UpLink are trademarks of RadiSys Corporation.

† All other trademarks, registered trademarks, service marks, and trade names are the property of their respective owners.

Contents

: Contents

Chapter 1: Installing and Configuring OS-9®

Development Environment Overview	8
Requirements and Compatibility	9
Host Hardware Requirements (PC Compatible)	9
Host Software Requirements (PC Compatible)	9
Target Hardware Requirements	9
Target Hardware Setup	10
Switch Settings	10
Connecting the Target to the Host	11
Building the ROM Image	15
Coreboot	15
Bootfile	15
Starting the Configuration Wizard	16
Creating the ROM Image	17
Creating the Coreboot Image	18
Creating the Bootfile Image	18
Building a ROM Image.....	18
Bootstrapping OS-9	21
Burning the Flash Part	23
Building a ROM Image for the pflash Utility	23

Chapter 2: Board-Specific Reference

Boot Options	30
Booting from Flash	30
Booting over a Serial Port via kermit.....	31
Restart Booter	31
Break Booter	31
Sample Boot Session and Messages	31
The Fastboot Enhancement.....	32
Overview	32
Implementation Overview	33

B_QUICKVAL.....	33
B_OKRAM	33
B_OKROM.....	34
B_1STINIT	34
B_NOIRQMASK	34
B_NOPARITY.....	34
Implementation Details	35
Compile-time Configuration	35
Runtime Configuration.....	35
OS-9 Vector Mappings	36
Fast Interrupt Vector (0x7).....	38
Port Specific Utilities	39
dmppci.....	40
pcip_host	41
pcip_option	42
pcip_test	43
pciv.....	44
pflash.....	45
setpci	46
Appendix A: Board-Specific Modules	
Low-Level System Modules	48
High-Level System Modules.....	49
CPU Support Modules.....	49
System Configuration Module	49
Interrupt Controller Support	50
Ticker	50
Generic I/O Support Modules (File Managers).....	50
Pipe Descriptor	51
RAM Disk Support	51
RAM Descriptors	51
Serial and Console Devices.....	51
Descriptors for use with scixc1100	52
Descriptors for use with scllio	52
SPF Device Support	53
Support for NPE ports.....	53
spethix Descriptors	53
Network Configuration Modules	53
Port Specific Utilities	53

Common System Modules List	54
----------------------------------	----

1

Installing and Configuring OS-9®

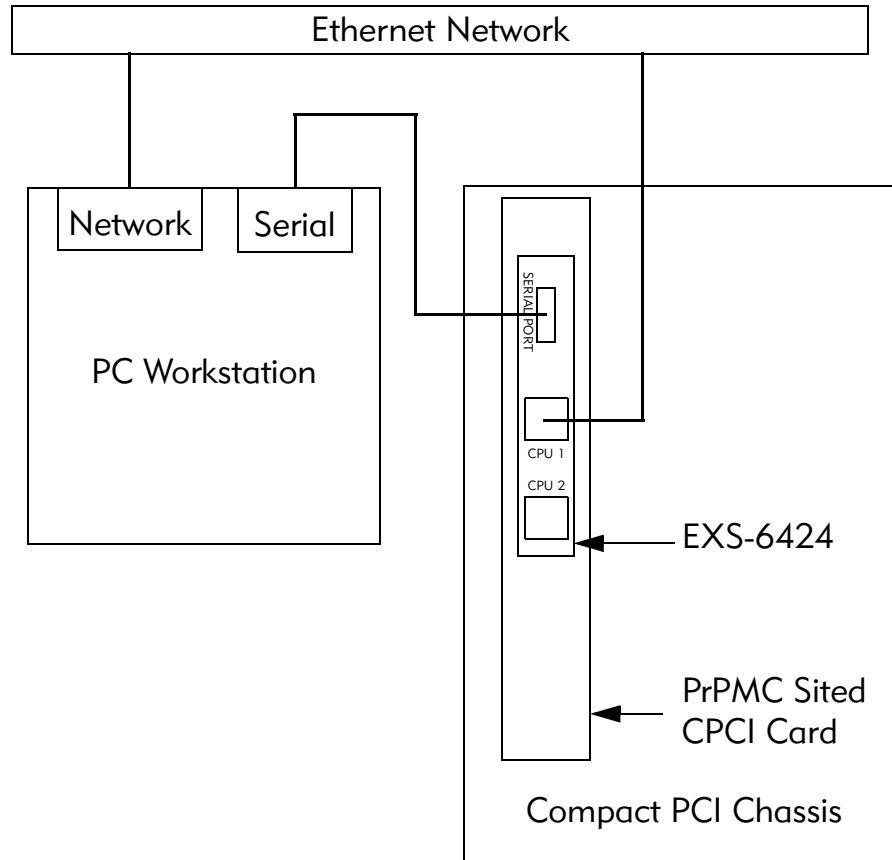
This chapter describes how to install and configure OS-9 on the RadiSys EXS-6424 Dual Embedded XScale PrPMC Board. It includes the following sections:

- [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 RadiSys EXS-6424. The components shown are the minimum required to develop software with OS-9 and the RadiSys EXS-6424.

Figure 1-1. EXS-6424 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.

- FTP client and server. The server is required to bootstrap OS-9 onto the EXS-6424. Once an initial bootable OS-9 image is programmed into the Flash, updating the boot image requires only an FTP client or kermit download facility.

Target Hardware Requirements

Your reference board requires the following hardware:

- Compact PCI enclosure or chassis with power supply and backplane
- Compact PCI card with PrPMC site

- Special RS-232 serial cable with three 9-pin connectors (product code 9P-SER-BRKOUT-CBL)
- Access to an Ethernet network

Target Hardware Setup

Switch Settings

Verify that the switch settings on the board match the following factory settings:

- Switch Block SW2, switch 1: OFF
- Switch Block SW2, switch 2: OFF

These settings specify that neither processor swaps the 8MB halves of its Flash part.

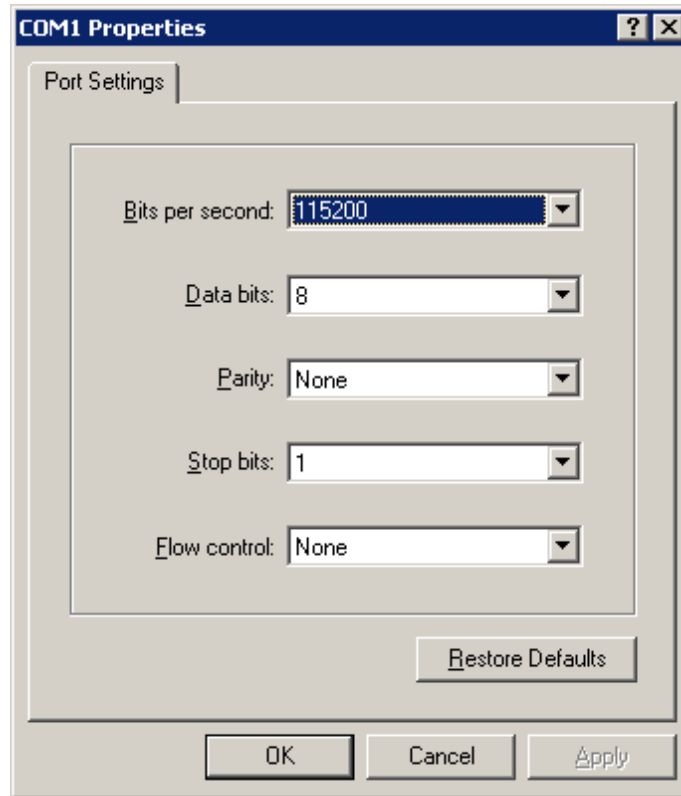
Connecting the Target to the Host

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

- Step 1. Install your EXS-6424 PrPMC board into your Compact PCI board's PrPMC site.
- Step 2. Connect the chassis to a power supply. Ensure that the power switch is in the OFF position.
- Step 3. Connect the EXS-6424 to an Ethernet network. For a detailed view, see [Figure 1-1](#). Use the front-panel RJ-45 connector marked "CPU 1".
- Step 4. Plug the miniature 9-pin connector of the special serial cable into the EXS-6424 and plug the normal 9-pin connector representing CPU #1 into the host system. For a detailed view, see [Figure 1-1](#).
- 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. When the countdown starts, press the space bar to stop the auto-boot. Output similar to the following displays:

```

ixe 0 Version information
Major ID:      0x0
Minor ID:      0x0
Build ID:      0x31
ixe 1 Version information
Major ID:      0x0
Minor ID:      0x0
Build ID:      0x31
ixe ETH PHY 0 MAC address is: 00:00:50:12:24:b8

ixe ETH PHY 1 Autonegotiate failed: Disabling
ixe ETH PHY 1 MAC address is: 00:00:50:12:24:b9
  
```

VxWorks System Boot

Copyright 1984-2002 Wind River Systems, Inc.

```
CPU: Radisys EXS6424 IXC1100 PrPMC (Big Endian)
Version: VxWorks5.5
BSP version: 1.0/0
Creation date: Oct  2 2003, 16:55:56
```

Press any key to stop auto-boot...

10

[VxWorks Boot]:

Step 11. Record the PHY 0 MAC address (00:00:50:12:24:b8 in this example) for use in the [Building the ROM Image](#) section.

Step 12. Use the `c` command at the **VxWorks Boot:** prompt to change the boot parameters to correct values for your network and FTP server.

[VxWorks Boot]: `c`

'.' = clear field; '-' = go to previous field; ^D = quit

boot device : `ixe0 <cr>`

Leave the boot device set to `ixe0` to use the front-panel Ethernet interface for the FTP download.

processor number : `1 <cr>`

Leave the processor number set to one.

host name : `testserver ftpserver`

Set this to the host name of the machine running your FTP server. This does not have to be the same machine used as the serial console.

file name : `/home/iapc/ftp/exs6424/vxworks rom.elf`

`rom.elf` is the file that downloads from the FTP server. If you cannot place `rom.elf` at the "root" of the FTP server's file structure, include the pathlist required to reach a suitable directory.

inet on ethernet (e) : `10.128.180.87 192.168.10.1`

Specify the IP address the EXS-6424 target plans to use. If required, you can specify a netmask by adding a colon and using hex notation for it (e.g. 192.168.10.1:ffffff00).

```
inet on backplane (b) : <cr>
```

Leave the IP address for the backplane Ethernet interface empty.

```
host inet (h) : 10.100.2.15 192.168.10.2
```

Specify the IP address of your FTP server machine.

```
gateway inet (g) : 10.128.180.250 192.168.10.254
```

Specify your network gateway's IP address, if any.

```
user (u) : iapc os9
```

Specify the user name used to log into your FTP server to download the rom.elf file.

```
ftp password (pw) (blank = use rsh) : iapctest os9
```

Specify the password for the above username.

```
flags (f) : 0x0 <cr>
```

There are no applicable flags. To see the list of possible flags, enter the he (help) command at the **VxWorks Boot:** prompt.

```
target name (tn) : M1 os9exs
```

Specify the EXS-6424 target's network host name.

```
startup script (s) : <cr>
```

Leave startup script empty.

```
other (o) : <cr>
```

Specify that there is no other information. After pressing <cr> this last time, the system briefly pauses while permanently storing on the target system the information you entered.

Your target is now ready to download the bootstrap boot image from the FTP 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.

The EXS-6424 has these different system types:

- Normal Core: This coreboot is configured to be programmed into the Flash part at the lowest address (0x50000000). It can boot OS-9 upon powerup.
- High Core: This coreboot is configured to be programmed into the second 8MB of the 16MB Flash part (0x50800000). It can facilitate fail-safe boot updates because it can be programmed into an unused area of the Flash and tested without endangering the current working boot.
- RAM Boot Core: This coreboot is configured to be loaded via the VxWorks boot monitor into RAM at address 0x10200000.

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.

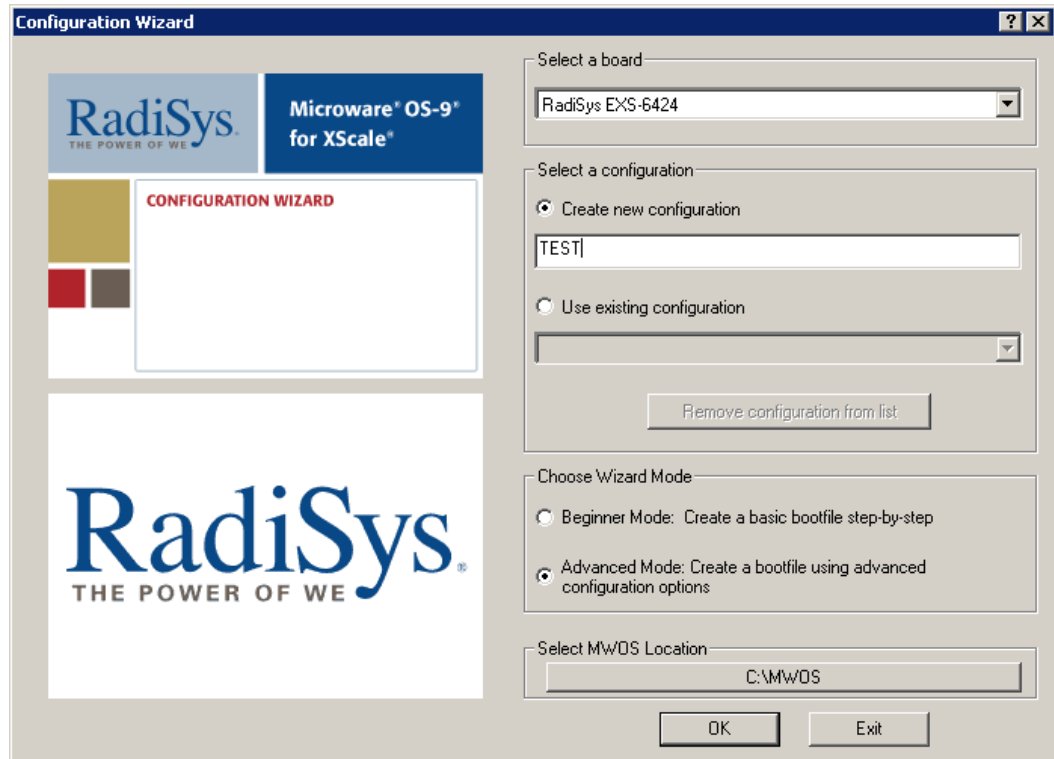
Microware provides a Configuration Wizard to create a coreboot image, a bootfile image, or an entire OS-9 ROM Image. The wizard can also modify an existing image. The Configuration Wizard automatically installs on your host PC during the installation process.

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**.

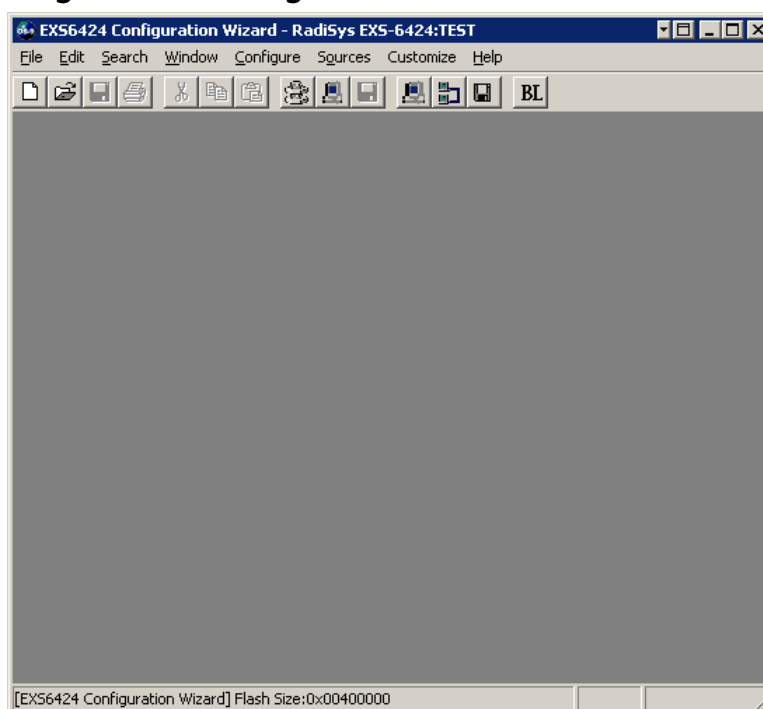
Figure 1-3. Configuration Wizard Opening Screen



- Step 2. Select your target board, RadiSys EXS-6424, 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 next 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 RadiSys EXS-6424 coreboot configuration is valid and does not require modification to create a working system. The default system type is RAM Boot Core, since the initial boot loads into RAM by the VxWorks monitor.

Creating the Bootfile Image

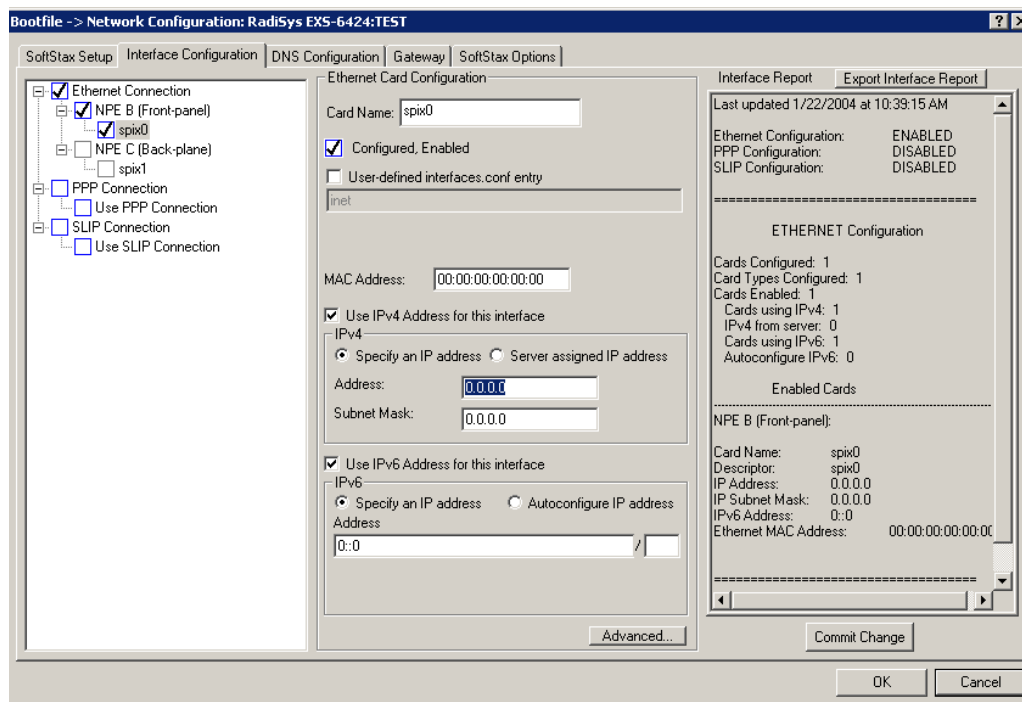
The default settings in the Configuration Wizard are preset for optimum performance for the RadiSys EXS-6424 Embedded XScale board. The only modifications required are to enable networking and to change the network settings. The network settings information must be obtained from your network administrator.

Building a ROM Image

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. Also, the MAC address should be set. Use the MAC address printed when the board boots upon powerup. The next figure shows an example of the **Interface Configuration** tab.

Figure 1-5. Network Configuration Dialog



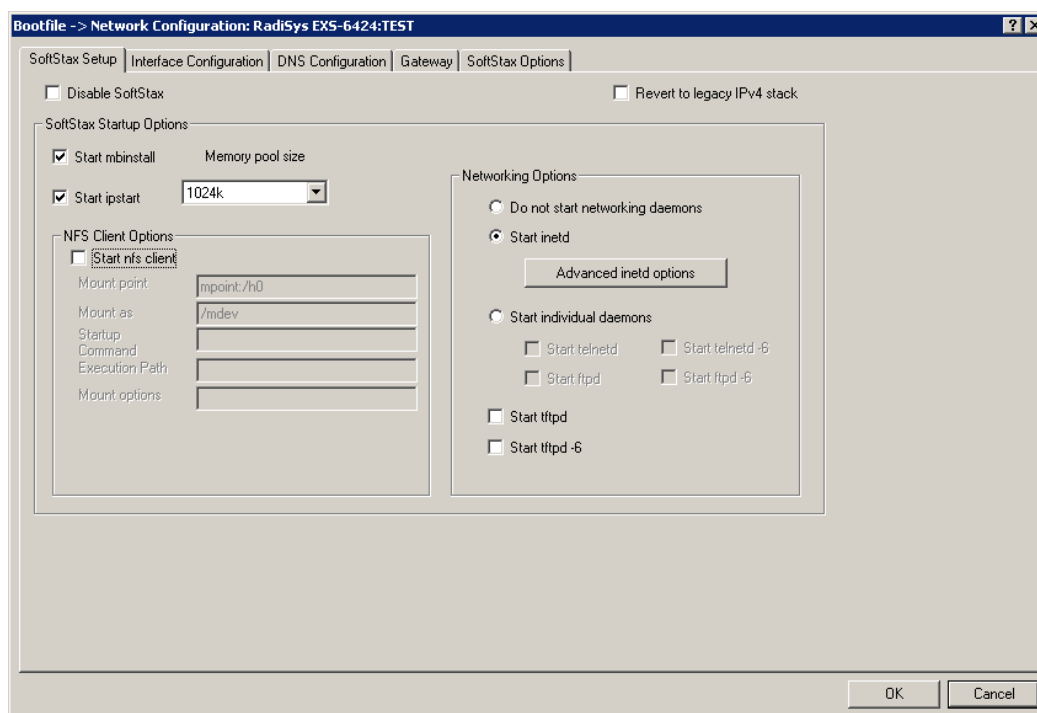
To learn more about IPv4 and IPv6 functionalities, see the *Using LAN Communications* manual included with this product CD.



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

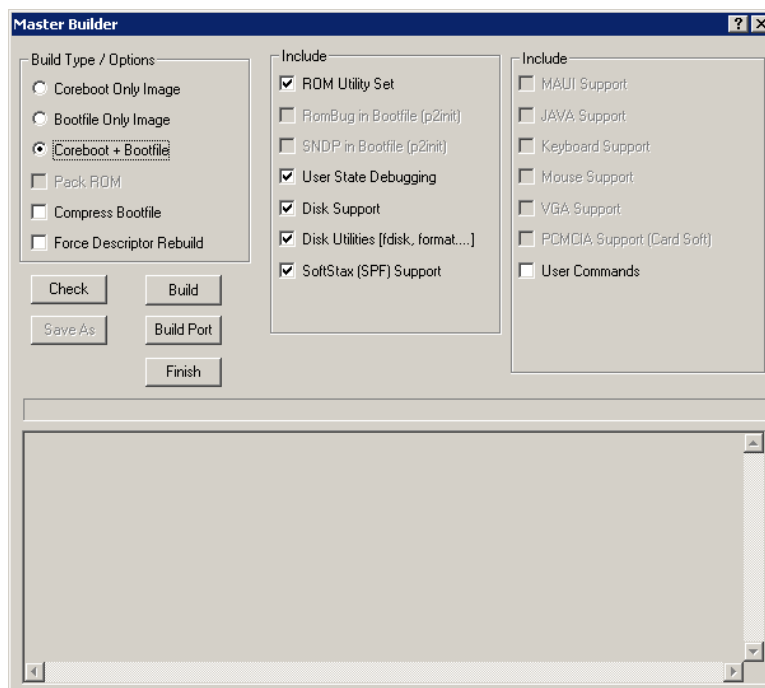
- Step 3. Select the **SoftStax® Setup** tab. The **SoftStax Setup** dialog window displays as shown in the next figure.

Figure 1-6. SoftStax Setup Dialog



- Step 4. Configure your system as shown in the previous figure.
- Step 5. Leave the other **Network Configuration** options at the default settings. Click **OK**.
- Change other network configuration options in this dialog, such as DNS and gateway, according to your specific requirements and network.
- Step 6. Select **Configure** -> **Build Image**. The **Master Builder** dialog window displays as shown in the next figure.

Figure 1-7. Master Builder Dialog



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

Step 8. Click **Build**. This builds a ROM image that the VxWorks boot monitor can load. The image, `rom.elf`, is stored in the following directory:

```
\MWOS\OS9000\ARMV5\PORTS\EXS6424\BOOTS\INSTALL\PORTBOOT
```

Copy the `rom.elf` file from this directory to the correct location for your FTP server.

Bootstrapping OS-9

The target is now configured, the `rom.elf` file is copied into place, and you are ready to boot OS-9 on your EXS-6424.

Step 1. Power up the chassis or press the reset button on the EXS-6424's front panel.

Step 2. Allow the auto-boot to time out.



Alternatively, you can abort the auto-boot delay and issue the `@` command at the **[VxWorks Boot]:** prompt.

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

```
ixe 0 Version information
Major ID:          0x0
Minor ID:          0x0
Build ID:          0x31
ixe 1 Version information
Major ID:          0x0
Minor ID:          0x0
Build ID:          0x31
ixe ETH PHY 0 MAC address is: 00:00:50:12:24:b8

ixe ETH PHY 1 Autonegotiate failed: Disabling
ixe ETH PHY 1 MAC address is: 00:00:50:12:24:b9
```

VxWorks System Boot

Copyright 1984-2002 Wind River Systems, Inc.

```
CPU: Radisys EXS6424 IXC1100 PrPMC (Big Endian)
Version: VxWorks5.5
BSP version: 1.0/0
Creation date: Oct 2 2003, 16:55:56
```

Press any key to stop auto-boot...

0

auto-booting...

```
boot device          : ixe
unit number          : 0
processor number     : 1
host name            : ftpserver
file name            : rom.elf
inet on ethernet (e) : 192.168.10.1
```

```
host inet (h)           : 192.168.10.2
gateway inet (g)       : 192.168.10.254
user (u)               : os9
ftp password (pw)     : os9
flags (f)              : 0x0
target name (tn)      : os9exs
```

```
Attached TCP/IP interface to ixex0.
Attaching network interface lo0... done.
Loading... 1989272
Starting at 0x10200000...
```

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

```
Now trying to Override autobooters.
```

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

```
Now trying to Copy embedded OS-9 to RAM and boot.
Compressed bootfile found at $10300000
A valid OS-9 bootfile was found.
+3
+6
$
```

Burning the Flash Part

You will need to burn a ROM file into the Flash parts. This is done by changing the ROM core configuration type and programming the created ROM image into the Flash.

Building a ROM Image for 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. Change the coreboot configuration by selecting the main menu item `Configure -> Select System Type...` Click the `Normal Core` radio button in the **Romcore type** group. Click `OK`.

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

Step 3. Select `Configure -> Build Image..` to display the **Master Builder** screen.

Step 4. Ensure `Coreboot + Bootfile` is selected and click `Build`.

Step 5. 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\EXS6424\BOOTS\INSTALL\PORTBOOT
```

Step 6. Start a DOS shell on the host system.

Step 7. Navigate to the directory in which the OS-9 ROM image, `rom`, is located.

Step 8. 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 9. 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 10. Log in with the username `super` and the password `user`.

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

```
ftp> cd /r1
```

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

Step 12. Enter the following command:

```
ftp> bin
```

This designates binary format.

Step 13. 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 14. On the target, enter the following command:

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

The file is programmed into the target system's Flash memory. `pflash` is configured to copy the `rom` file to the correct address (`0x50000000`). Normally, this address is protected because it holds the fail-safe ROM image; the `-np` option overrides the protection mechanism. This operation overwrites the VxWorks boot monitor with a bootable version of OS-9. The next time the board is reset or power cycled, it will boot directly into OS-9.



If, for some reason, OS-9 does not boot properly or the network settings do not allow for connectivity, the VxWorks boot monitor can be recovered. To reprogram the OS-9 Flash image, follow these steps:

1. Power the board off and change the position of switch SW2,1 from OFF to ON. This swaps the two halves of the 16MB Flash part for CPU #1. What was once at `0x50000000` is now at `0x50800000` and vice versa.
2. Use the preceding procedure to boot OS-9 from RAM, using the already existing `rom.elf`.
3. Fix the OS-9 boot in whatever manner required. Continue to build a Normal Core ROM configuration.
4. Download the new `rom` file, but program it into the Flash using a different command:


```
$ pflash -s=50800000 -f=/r1/rom
```
5. Once the programming is completed, power the board off and return the switch SW2,1 to the OFF position and power up the board.

Now that a working OS-9 boot is in the Flash in the 0x50000000 – 0x50800000 space and switch SW2,1 is in the OFF position, new boots can be tested by:

- Creating boots using the High Core system type in the Configuration Wizard.
- Programming boots into the upper half of the Flash part (0x50800000).
- Re-creating boots with the Normal Core system type in the Configuration Wizard.
- Reprogramming boots at the base of the Flash.

The following steps can be used to validate a new boot:

- Step 1. Create the new boot using the Configuration Wizard, making sure to use the High Core ROM system type.
- Step 2. Transfer the `rom` file to the OS-9 target via FTP as described previously.
- Step 3. Reprogram the upper half of the Flash part.
- Step 4. Use the ROM debugger to call the new boot to ensure it boots.

```
$ pflash -f=/r1/rom -s=50800000

$ break
<Called>
r0 :60000013  r1 :00000000  r2 :00000000  r3 :00000000  r4 :00000000
r5 :00000000  r6 :10932FB0  r7 :00000000  r8 :00000000  r9 :10608C00
r10: 00000001 r11:1BD69A3C r12:1093AB54  sp :1BD69A2C lr :1063E5B8
pc: 1063E5B8 cpsr: 60000093 (-ZC-----I---10011)
0x1063E5B8 >E1A07000          mov    r7,r0
RomBug: g 50800000
```

OS-9 Bootstrap for the ARM (Edition 68)

Now trying to Override autobooters.

Press the spacebar for a booter menu

Now trying to Copy embedded OS-9 to RAM and boot.

Compressed bootfile found at \$50900000

A valid OS-9 bootfile was found.

+3

+4

\$

- Step 5. Change the system type to Normal Core via the **Configure** -> **Select System Type..** Configuration Wizard menu.
- Step 6. Reprogram the upper half of the Flash part using the command from Step 3.
- Step 7. Change the position of switch SW2,1 to the opposite position. Now, the old working boot becomes the upper half of the Flash and can be overwritten with newer boots using this same procedure.



Once you complete this procedure, the VxWorks boot monitor no longer exists in the Flash part. Be careful to avoid a situation where both halves of the Flash contain non-working boots.

2

Board-Specific Reference

This chapter contains porting information specific to the RadiSys EXS-6424 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 RadiSys EXS-6424 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\EXS6424\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_cfg.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_cfg.h` is located in the following directory:

```
MWOS\OS9000\ARMV5\PORTS\EXS6424\ROM\ROMCORE
```

<code>bo</code>	ROM boot—the system runs from the Flash bank. You cannot use this booter with an uncompressed boot since the EXS-6424's Flash memory is not byte addressable.
<code>lr</code>	load to RAM—the system copies the Flash 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 a RadiSys EXS-6424 example boot using the bo boot option.

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

```
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: bo

Compressed bootfile found at $50900000
A valid OS-9 bootfile was found.
+3
$ mfree
Current total free RAM: 178696.00 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 *gblbs)
{
    /* if switch or jumper setting is set... */
    if (switch_or_jumper == SET) {
        /* force checking of ROM and RAM lists */
        rinf->os->boot_config &= ~(B_OKROM+B_OKRAM);
    }
    return SUCCESS;
}

```

OS-9 Vector Mappings

This section contains the OS-9 vector mappings for the RadiSys EXS-6424 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 RadiSys EXS-6424

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. RadiSys EXS-6424 Specific IRQ Assignments

OS-9 IRQ #	EXS-6424 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	Inter-processor From Other CPU
0x47	GPIO[1]
0x48	PCI Interrupt
0x49	PCI DMA Channel 1
0x4a	PCI DMA Channel 2
0x4b	General Purpose Timer 1
0x4d	Console UART (Front-panel)
0x4e	Timestamp Timer
0x4f	High-speed UART (Back-plane)
0x50	Watchdog Timer
0x51	Performance Monitoring Unit Counter Rollover
0x52	XScale PMU Counter Rollover
0x53	GPIO[2]
0x54	GPIO[3]
0x55	Port 0 PHY
0x56	Port 1 PHY
0x57	GPIO[6]
0x58	GPIO[7]
0x59	PMC INTD (Monarch CPU 1 Only)
0x5a	PMC INTC (Monarch CPU 1 Only)
0x5b	PMC INTB (Monarch CPU 1 Only)
0x5c	PMC INTA (Monarch CPU 1 Only)
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 re-define a source as an FIQ to make use of this feature.

Port Specific Utilities

Utilities for the RadiSys EXS-6424 are located in the following directory:

MWOS/OS9000/ARMV5/PORTS/EXS6424/CMDS

The following port specific utilities are included:

<code>dmppci</code>	shows PCI device information
<code>pcip_host</code>	facilitates communication between the two IXC1100 processors from the host side
<code>pcip_option</code>	facilitates communication between the two IXC1100 processors from the option side
<code>pcip_test</code>	tests the PCI pipes functionality
<code>pciv</code>	displays board PCI bus information
<code>pflash</code>	programs onboard Flash
<code>setpci</code>	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.

pcip_host Facilitates Communication Between Processors

Syntax

`pcip_host [options]`

Options

`-z[[=]<file>]` read additional command line arguments from <file> (default = standard input)

Description

`pcip_host` facilitates communication between the two IXC1100 processors via the PCI shared memory area. This program should be executed only on the PCI host processor (Processor 1 when the PrPMC card is in Monarch mode). It creates two named pipes:

- `/pipe/pci_out` for information destined for the other processor.
- `/pipe/pci_in` for information from the other processor.

`pcip_host` then continues to run in the background, forwarding information to and from the other processor.

pcip_option Facilitates Communication Between Processors

Syntax

`pcip_option [options]`

Options

`-z[[=]<file>]` read additional command line arguments from <file> (default = standard input)

Description

`pcip_option` facilitates communication between the two IXC1100 processors via the PCI shared memory area. This program should be executed only on the PCI option (non-host) processor (Processor 2 when the PrPMC card is in Monarch mode). It creates two named pipes:

- `/pipe/pci_out` for information destined for the other processor.
- `/pipe/pci_in` for information from the other processor.

`pcip_option` then continues to run in the background, forwarding information to and from the other processor.

pcip_test

Tests the PCI Pipes Functionality

Syntax

```
pcip_test [<opts>] {<file>}
```

Options

-v	verbose mode
-z[[=]<file>]	read additional command line arguments from <file> (default = standard input)

Description

`pcip_test` transfers files from one processor to the other.

Example

This is just a simple example of how to use the `pcip_host` and `pcip_option`.

For this simple demonstration:

1. Specify one processor as the server and the other processor is then a client.
2. On the server, execute `pcip_test` in the background (e.g. `pcip_test &`).
3. On the other processor, use `pcip_test` to transfer files (e.g. `pcip_test testfile1 testfile2`).

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 0x50800000.

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 RadiSys EXS-6424. 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/EXS6424/CMDS/BOOTOBS/ROM

<code>cnfgdata</code>	Contains the low-level configuration data.
<code>cnfgfunc</code>	Provides access services to <code>cnfgdata</code> data.
<code>commcnfg</code>	Initializes the communication port defined in <code>cnfgdata</code> .
<code>conscnfg</code>	Initializes the console port defined in <code>cnfgdata</code> .
<code>initext</code>	Performs basic initialization of the PCI sub-system.
<code>io16550</code>	Provides low-level serial services via the on-board UARTs.
<code>pciwalk</code>	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.
<code>portmenu</code>	Initializes booters defined in <code>cnfgdata</code> .
<code>romcore</code>	Provides board-specific initialization code. Place at the Flash base address (0x50000000).
<code>romcore_hi</code>	Provides board-specific initialization code. Place at the Flash base address (0x50800000).
<code>romcore_ram</code>	Provides board-specific initialization code. The VxWorks boot monitor loads the code into RAM at address 0x10200000 and executes from there.
<code>tmrxc1100</code>	Uses general purpose timer 0 to provide timer services for the low-level system.
<code>usedebug</code>	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 RadiSys EXS-6424. Unless otherwise specified, each module is located in a file of the same name in the following directory:

MWOS/OS9000/ARMV5/PORTS/EXS6424/CMDS/BOOTOBS

CPU Support Modules

These files are located in the following directory:

MWOS/OS9000/ARMV5/CMDS/BOOTOBS

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

System Configuration Module

The system configuration modules are located in the following directory:

MWOS/OS9000/ARMV5/PORTS/EXS6424/CMDS/BOOTOBS/INITS

<code>dd</code>	High-level initialization module that specifies <code>/dd</code> as the initial disk device.
<code>nodisk</code>	High-level initialization module that specifies no initial disk device. This <code>init</code> module is used in diskless systems.
<code>configurer</code>	<code>init</code> 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.

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

Ticker

<code>tkixc1100</code>	Driver that provides the system ticker based on the IXC1100's general purpose timer 1.
<code>hcsub</code>	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/CMD5/BOOTOBS`

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

Pipe Descriptor

The pipe descriptor is located in the following directory:

MWOS/OS9000/ARMV5/PORTS/EXS6424/CMDS/BOOTOBS/DESC

<code>pipe</code>	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/BOOTOBS

<code>ram</code>	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/EXS6424/CMDS/BOOTOBS/DESC/RAM

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

Serial and Console Devices

<code>scixc1100</code>	SCF driver that provides serial support the IXC1100's internal UARTs.
------------------------	---

Descriptors for use with `scixc1100`

<code>term1/t1</code>	Descriptor modules for use with <code>scixc1100</code> Front-panel UART Default Baud Rate: 115200 Default Parity: None Default Data Bits: 8 Default Stop Bits: 1 Default Handshake: XON/XOFF
<code>term2/t2</code>	Descriptor modules for use with <code>scixc1100</code> Backplane UART Default Baud Rate: 115200 Default Parity: None Default Data Bits: 8 Default Stop Bits: 1 Default Handshake: XON/XOFF
<code>scllio</code>	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\EXS6424\CMD5\BOOTOBJS\DESC\SCLLIO
```

<code>vcons/term</code>	Descriptor modules for use with <code>scllio</code> in conjunction with a low-level serial driver. Port configuration and setup follows that which is configured in <code>cnfgdata</code> for the console port. <code>scllio</code> can communicate with a true low-level serial device driver like <code>io16550</code> , or with an emulated serial interface provided by <code>iovcns</code> .
-------------------------	---



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/EXS6424/CMDS/BOOTOBS/SPF

`spethix` SPF driver to support ethernet via NPE B (front-panel) or NPE C (back-plane).

spethix Descriptors

These descriptor files are located in the following directory:

MWOS/OS9000/ARMV5/PORTS/EXS6424/CMDS/BOOTOBS/SPF

`spix0` SPF descriptor module for use with the front-panel Ethernet connector.

`spix1` SPF descriptor module for use with backplane Ethernet connector.

Network Configuration Modules

These files are located in the following directory:

MWOS/OS9000/ARMV5/PORTS/EXS6424/CMDS/BOOTOBS/SPF

`inetdb` Internet database information.

`inetdb2` Target-specific database information.

`rpcdb` RPC database information.

Port Specific Utilities

The following EXS-6424-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/EXS6424/CMDS

`dmppci` Allows a specific PCI device's configuration area to display. This should be executed only on a PCI host device.

<code>pcip_host</code>	Facilitates communication between the two IXC1100 processors via the PCI shared memory area on the host side.
<code>pcip_option</code>	Facilitates communication between the two IXC1100 processors via the PCI shared memory area on the option side.
<code>pcip_test</code>	Transfers files from one processor to the other.
<code>pciv</code>	Displays configuration information about all available PCI devices. This should execute only on a PCI host device.
<code>pflash</code>	Programs the on-board Intel StrataFlash device.
<code>setpci</code>	Allows changes to the PCI configuration of devices. This should be executed only on a PCI host device.

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/CMD5/BOOTOBSJ5/ROM`

<code>bootsys</code>	Provides booter registration services.
<code>console</code>	Provides console services.
<code>dbgentry</code>	Inits debugger entry point for system use.
<code>dbgserv</code>	Provides debugger services.
<code>exception</code>	Provides low-level exception services.
<code>flshcach</code>	Provides low-level cache management services.
<code>hlproto</code>	Provides user-level code access to <code>protoman</code> .
<code>llbootp</code>	Provides bootp services.
<code>llip</code>	Provides low-level IP services.
<code>llslip</code>	Provides low-level SLIP services.
<code>lltcp</code>	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*.

