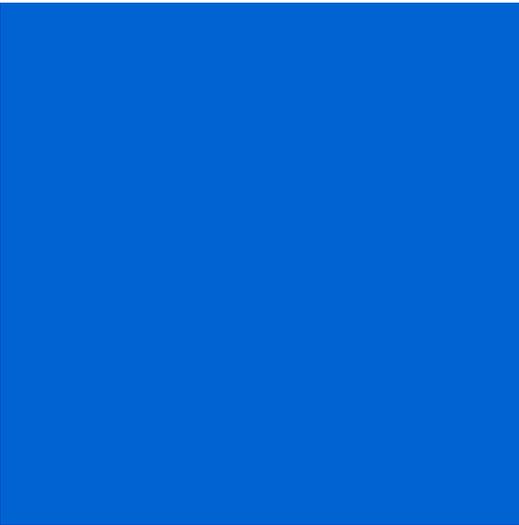


[Home](#)



# **OS-9<sup>®</sup> for Toshiba RBHM4x00 Board Guide**

## **Version 4.7**



**RadiSys.**  
THE POWER OF WE

[www.radisys.com](http://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

---

## Installing and Configuring OS-9® 5

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 and Host Setup.....	7
Settings.....	8
Switch Configuration to Run OS-9 or Toshiba HCP5 Boot Monitor.....	8
Switch Configuration to Run OS-9.....	8
Switch Configuration to Run HCP5 Monitor.....	8
Connecting the Target to the Host.....	8
Attaching the Cables.....	8
Booting to the Boot Menu.....	9
Building the OS-9 ROM Image.....	10
Coreboot.....	10
Bootfile.....	10
Starting the Configuration Wizard.....	11
Configuring Coreboot and Bootfile Options.....	12
Ethernet Configuration (Coreboot).....	12
Ethernet Configuration (Bootfile).....	13
Building the Coreboot + Bootfile Image.....	14
Transferring the ROM Image to the Target.....	14
Optional Procedures.....	16
Programming a ROM Image with the pflash Utility.....	16
Flashed ROM Image Issues.....	17
Building with Makefiles.....	18
Makefile Network Option.....	18
Using Makefiles.....	18
Making Network Configuration Changes.....	19
Low-Level Network Configuration Changes.....	19

## Board Specific Reference 21

Boot Options.....	22
Booting from Flash.....	22
Booting over a Serial Port via kermit.....	22
Restart Booter.....	22
Break Booter.....	23
Sample Boot Session and Messages.....	23
The Fastboot Enhancement.....	24
Overview.....	24
Implementation Overview.....	24
B_QUICKVAL.....	24

B_OKROM .....	25
B_1STINIT .....	25
B_NOIRQMASK .....	25
B_NOPARITY .....	25
Implementation Details.....	25
Compile-time Configuration.....	26
Runtime Configuration .....	26
OS-9 Vector Mappings.....	27
Port Specific Utilities.....	29
dmppci .....	30
pciv .....	31
pflash .....	32
setpci .....	33
<b>Board Specific Modules 35</b>	
Port-Specific Low-Level System Modules .....	36
Common Low-Level System Modules .....	36
Port-Specific High-Level System Modules.....	37
Port-Specific High-Level Utilities .....	38
Common High-Level System Modules .....	38

# 1

## Installing and Configuring OS-9®

---

This chapter describes installing and configuring OS-9® on the Toshiba RBHM4x00 boards. It includes the following sections:

[Development Environment Overview](#)

[Requirements and Compatibility](#)

[Target and Host Setup](#)

[Connecting the Target to the Host](#)

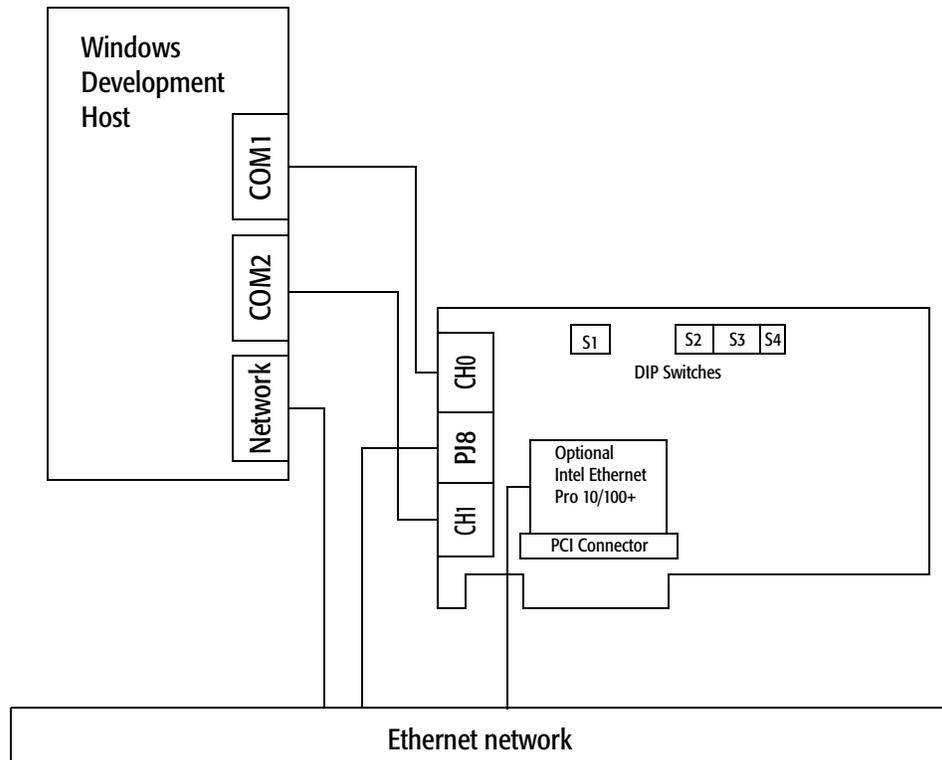
[Transferring the ROM Image to the Target](#)

[Optional Procedures](#)

## Development Environment Overview

Figure 1-1 shows a typical development environment for a Toshiba RBHM4x00 board. The components shown are those generally required to develop for OS-9 on the board.

Figure 1-1. Toshiba RBHM4x00 Development Environment



## Requirements and Compatibility

Before you begin, install the *Microware OS-9 for MIPS64* 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)

- Microware OS-9 for MIPS64
- 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.

- [Optional] BOOTP and TFTP servers. BOOTP and TFTP servers are required if Ethernet booting OS-9.

### Target Hardware Requirements

Your RBHM4200 or RBHM4400 boards requires the following hardware:

- a power supply
- an RS-232 null modem serial cable (for OS-9 console)

In addition to the above requirements, you may want to have a second RS-232 null modem serial cable for connecting to the Toshiba HCP5 monitor and an Ethernet cable and card. (Supported Ethernet card: Intel PRO 10/100.)

### Target and Host Setup



Before installing and configuring OS-9 on your RBHM4x00 evaluation board, refer to the hardware documentation for information on hardware setup.

## Settings

The factory default setting for the DIP switches may not work with OS-9. Initially, the Toshiba HCP5 boot monitor will be used. Be sure the DIP switches agree with the following settings:

### Switch Configuration to Run OS-9 or Toshiba HCP5 Boot Monitor

S1.1 set to OFF	Internal PCI Arbiter
S1.2 set to ON	Internal Timer Interrupts Disabled
S1.3-5 set to OFF, OFF, ON	Select 200MHz (4200)/300MHz (4400)
S1.6 set to OFF	Big Endian
S1.7 set to OFF	SYSCLK Divisor 3
S1.8 set to OFF	PICMG-compatible

### Switch Configuration to Run OS-9

S3.1,2 set to ON, OFF	FlashROM1 Connected to CEO
S3.3,4 set to OFF, ON	FlashROM2 Connected to CE1
S3.5,6 set to OFF, OFF	ROM Emulator Not Connected
S3.7,8 set to ON, OFF	32-bit Boot ROM Bus Width

### Switch Configuration to Run HCP5 Monitor

S3.1,2 set to OFF, ON	FlashROM1 Connected to CE1
S3.3,4 set to ON, OFF	FlashROM2 Connected to CEO
S3.5,6 set to OFF, OFF	ROM Emulator Not Connected
S3.7,8 set to OFF, ON	16-bit Boot ROM Bus Width

## Connecting the Target to the Host

Connecting the RBHM4x00 board to your host PC involves attaching the power, serial, and Ethernet cables to the reference board. Once you have the board connected, you can use the serial console in Hawk to verify the serial connection.

### Attaching the Cables

To attach the cables, complete the following steps:

1. Attach the power connector to PJ16.
2. Attach the serial cable to PJ20 (RS-232 CH1). (The HCP5 monitor is on PJ20; it will be used to load the flash image.)
3. Connect the optional serial cable to PJ19. (The default OS-9 console is on PJ19.) If you are only using one cable, you will need to switch it from PJ20 after the flash image is installed.

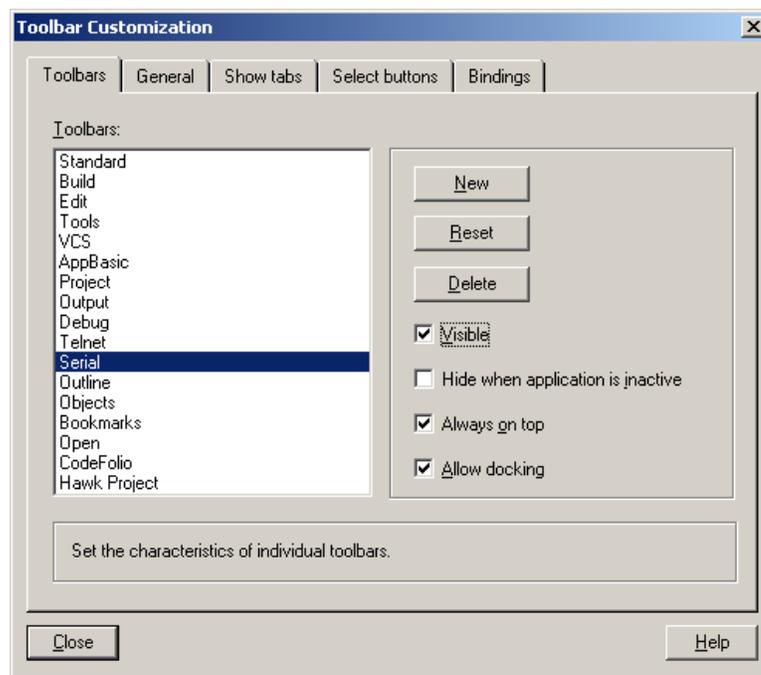
4. If you are using Ethernet, plug the cable into either the on-board RJ-45 (PJ8) or the RJ-45 on an Intel Ethernet Pro 10/100 in the PCI slot (PJ2).

## Booting to the Boot Menu

It will be necessary to boot to the HCP5 prompt in order to verify that your serial cable is connected properly. To do this, complete the following steps:

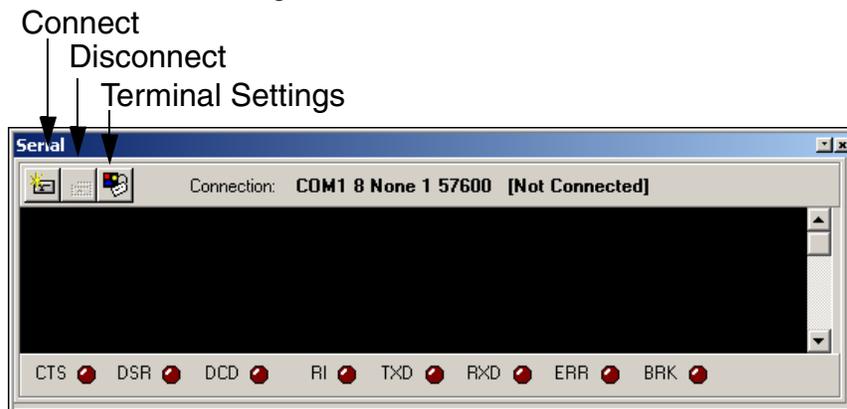
1. From the desktop, click Start and select **RadiSys -> Microware OS-9 for MIPS64 -> Hawk IDE** to start Hawk.
2. If the Serial console window is not open, it can be opened from the Toolbar Customization dialog (shown in [Figure 1-2](#)). (Select from the main menu **Customize -> Toolbars** to open the Toolbar Customization dialog.)

Figure 1-2. Toolbar Customization dialog



3. Once the dialog is open, select **Serial** in the Toolbars list box.
4. Click the **Visible** check box, then click the **Close** button. The Serial console window opens. (The Serial window can be seen in [Figure 1-3](#).)

Figure 1-3. Hawk Serial Console Window



5. Once you have the Serial Console window open, click the **Connect** button in the upper left corner of the window. The Com Port Options dialog box appears.
6. Change the setting to reflect these values: baud = 57,600, word size = 8 bits, parity = none, stop bits = 1, and XON/XOFF protocol.
7. Click the **OK** button. The message *[Not Connected]* should change to *[Connected]*.
8. Apply power to the board. The HCP5 boots the board.

## Building the OS-9 ROM Image

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

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

### Coreboot

The coreboot image is generally responsible for initializing hardware devices and locating the high-level image (AKA bootfile) as specified by its configuration. For example from a flash part, a harddisk, or Ethernet. It is also responsible for building basic structures based on the image it finds and passing control to the kernel to bring up the OS-9 system.

### Bootfile

The bootfile image contains the kernel and other high-level modules (initialization module, file managers, drivers, descriptors, applications). The image is loaded into memory based on the device you select from the boot menu. The bootfile image

normally brings up an OS-9 shell prompt, but can be configured to automatically start an application.

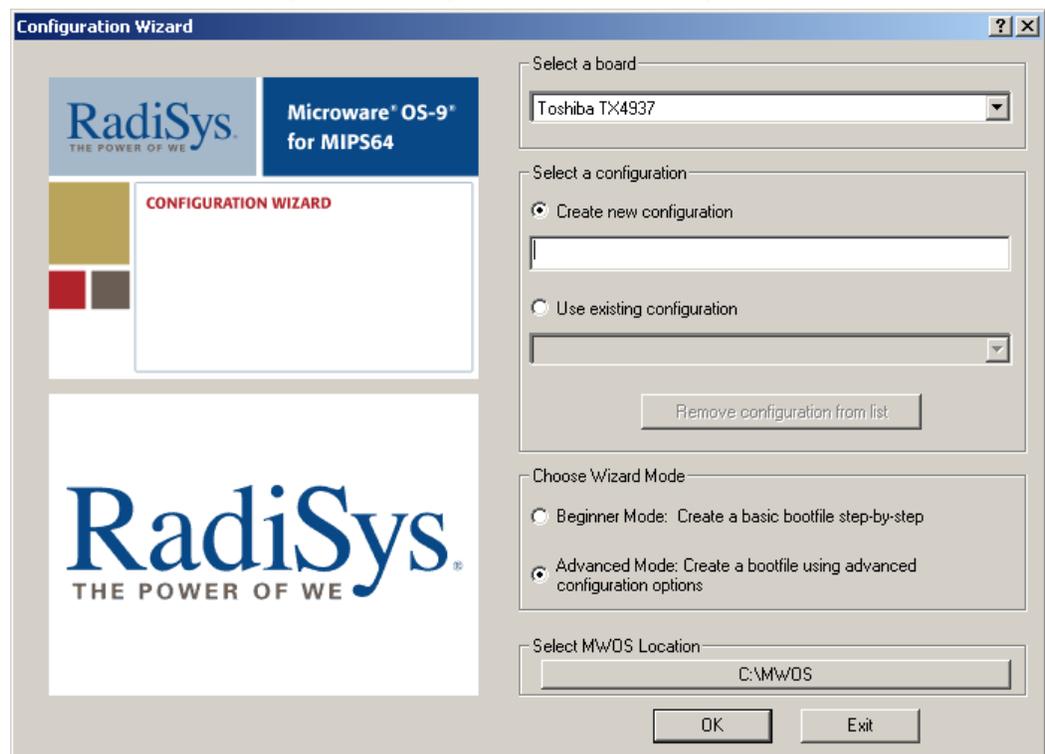
RadiSys provides a Configuration Wizard to create a coreboot image, a bootfile image, or an entire OS-9 ROM Image. The wizard can also be used to modify an existing image. The Configuration Wizard is automatically installed on your host PC during the OS-9 installation process.

## Starting the Configuration Wizard

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

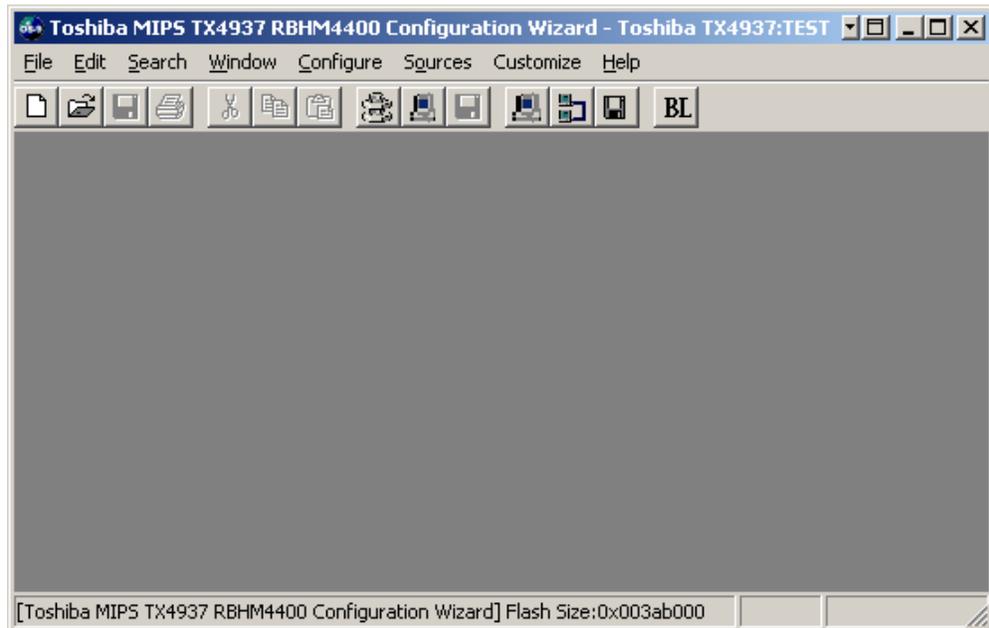
1. From the Windows desktop, select **Start -> RadiSys -> Microware OS-9 for MIPS64 -> Microware Configuration Wizard**. You should see the following opening screen:

Figure 1-4. Configuration Wizard Opening Screen



2. Select your target board from the Select a board pull-down menu.
3. Select the **Create new configuration** radio button from the Select a configuration menu and type in the name you want to give your ROM image in the supplied text box. This names your new configuration, which can later be accessed by selecting the Use existing configuration pull down menu.
4. Select the **Advanced Mode** radio button from the Choose Wizard Mode field and click **OK**. The Wizard's main window is displayed. This is the dialog from which you will proceed to build your image. An example is shown in [Figure 1-5](#).

Figure 1-5. Configuration Wizard Main Window



## Configuring Coreboot and Bootfile Options

The only steps necessary in configuring the coreboot and bootfile options involve filling in the Ethernet information. All other default settings in the Configuration Wizard are correct for the RBHM4x00 board.



If you do not have Ethernet installed on your system, you may skip the sections on Ethernet setup and proceed directly to the [Building the Coreboot + Bootfile Image](#) section.

### Ethernet Configuration (Coreboot)

To configure the Ethernet settings from the Configuration Wizard, complete the following steps:

1. From the Main Configuration window, select **Configure** -> **Coreboot** -> **Main configuration**.
2. Select the **Ethernet** tab.
3. Enter the Ethernet address information in the address text boxes. This includes the following information:
  - IP Address
  - Subnet Mask
  - IP Gateway

If you are uncertain of the values for these text boxes, contact your system administrator.

4. Select the appropriate Ethernet card in the drop down menu box at the bottom of the screen. (This should be either NE2000 or PCI Intel PRO 100.)

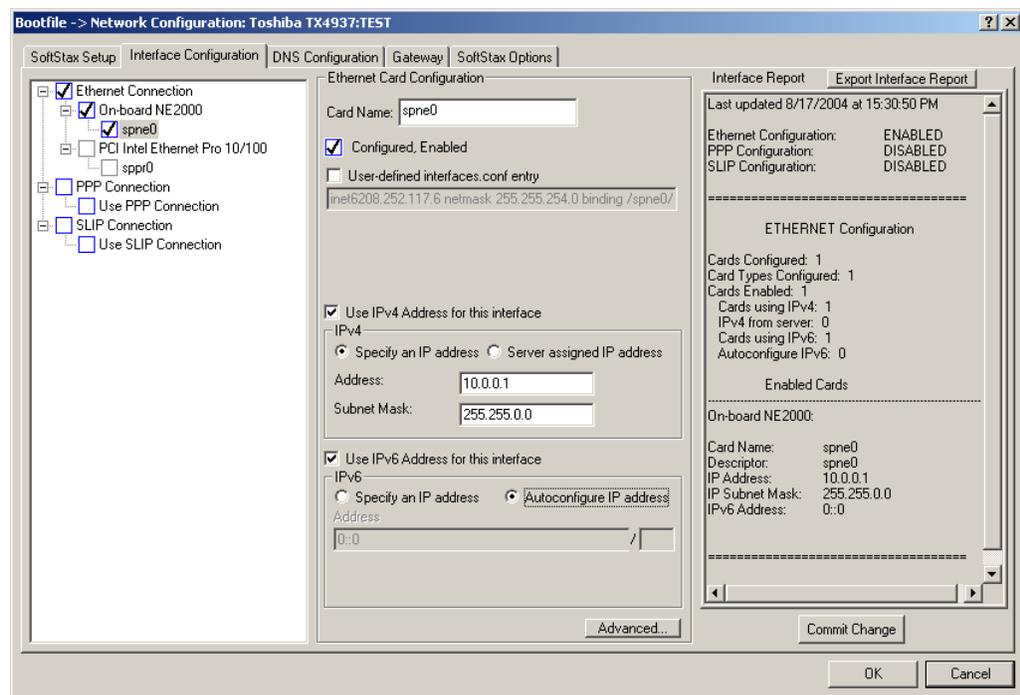
5. Check the Add to Boot Menu check box under the Ethernet Boot Options.
6. Click **OK** to close the window.

### Ethernet Configuration (Bootfile)

To configure the Ethernet settings from the Configuration Wizard, complete the following steps:

1. If you want to use the target board across a network, you will need to configure the Ethernet settings within the Configuration Wizard. To do this, select **Configure** -> **Bootfile** -> **Network Configuration** from the Wizard's main menu.
2. From the Network Configuration dialog, select the **Interface Configuration** tab. From here you can select and enable the interface. For example, you can select the appropriate Ethernet card from the list of options on the left and specify whether you would like to enable IPv4 or IPv6 addressing. [Figure 1-6](#) shows an example of the Interface Configuration tab.

Figure 1-6. Bootfile -> Network Configuration -> Interface Configuration



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



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

3. Select the **DNS Configuration** and **Gateway** tabs to specify setting specific to your site.
4. Select the **SoftStax Options** tab and select choose any networking options.
5. Click **OK** to close the dialog box.

## Building the Coreboot + Bootfile Image

Once your coreboot and bootfile settings have been properly configured, complete the following steps to build the ROM image:

1. Select **Configure** -> **Build Image**. The Master Builder window is displayed.
2. From the Master Builder window, select the check boxes appropriate for your setup:
  - If you are using a RAM disk, ensure the Disk Support and Disk Utilities check boxes are checked.
  - If you have Ethernet setup for your system, select the SoftStax (SPF) Support and User State Debugging Modules check boxes.
3. Select the **Coreboot + Bootfile** radio button.
4. Click **Build**. This builds the ROM image that can later be burned into flash memory. The name of this ROM image is `rom.s`. The file containing the image is in Motorola S-record format.
5. Click **Finish**.
6. Select **File** -> **Save Settings** to save the configuration.
7. Select **File** -> **Exit** to quit the Configuration Wizard.

## Transferring the ROM Image to the Target

In the previous section, you built a ROM image. To load this ROM image onto the target board, complete the following steps, using a capable terminal emulation program such as Hyperterminal:

1. Ensure switch bank 3 is set as specified earlier for running HCP5.
2. The serial cable should be connected to PJ20 (CH1) with a setting of 57,600, 8-bit words, no parity, and 1 stop bit. Once you apply power, the HCP5 prompt should appear.
3. At the HCP5 prompt, type `o`. The screen should display the following information:

```
HCP for RBTX4937 HCP5 Rev. 1.1.0 (big endian)
```

```
HCP5? o
Option menu
a: aclc
b: ether
c: flash
d: id
e: interrupt
f: led
```

```

g: pcic
h: rtc
i: sdram
j: serial
k: hcp5_csio_b4
Option command ?

```

4. At the Option command, type **c**. The following message appears:

```

Option command ? c 00000002 00007a48 a0020000 bfc2a800 a0020000
--

```

```

EBUSC0 adr=1f000000-20000000 EBCCR=01f00000 00203408 bus=16 ENA
EBUSC1 adr=1e000000-1f000000 EBCCR=01e00000 00103418 bus=32 ENA
EBUSC2 adr=1c000000-1c100000 EBCCR=01c00000 0033f018 bus= 8 ENA
EBUSC3 adr=10000000-14000000 EBCCR=01000000 0023f618 bus=16 ENA
EBUSC4 adr=00000000-00100000 EBCCR=00000000 00000000 bus= 0 DIS
EBUSC5 adr=00000000-00100000 EBCCR=00000000 00000000 bus= 0 DIS
EBUSC6 adr=00000000-00100000 EBCCR=00000000 00000000 bus= 0 DIS
EBUSC7 adr=00000000-00100000 EBCCR=00000000 00000000 bus= 0 DIS

```

HCP for RBTX4937 Flash ROM main menu:

```

d. Write boot ROM image to EBUSC1      r. Write ROM image
4. Copy boot ROM image to EBUSC1      t. Clear flash ROM
y. Simple access test                  i. Access test

x. Exit                                z. Go to main menu

```

command>

5. Type **d** to select the reprogramming of ROMCE1. The following message appears:

```

command>d

```

```

--- Writing S-record image to the ROM on EBUSC1 ---

```

```

convert endian?(say 'y' to write little endian binary) [y/n]

```

6. Type **n** to write a big endian binary. The following message appears:

```

convert endian?(say 'y' to write little endian binary) [y/n]n

```

```

ROM address be000000-bf000000, Boot Bus-[32bit]
Flash ROM-[TC58FVM6T2ATG65], [16bit chip] * 2 * 1
Block size-00020000H count-128 mask-00fe0000H
total-10000000H bytes (* 1)

```

```

Set Xon/Xoff, Send Srecord file

```

7. From the Hyperterminal window, transfer the `rom.s` file from the following directory to the target:

```
MWOS/OS9000/MIPS64/PORTS/RBHM4000/BOOTS/INSTALL/  
PORTBOOT
```

This transfer will take some time to complete. When it is finished the following messages are displayed:

```
Flash Write OK
```

HCP for RBTX4937 Flash ROM main menu:

```
d. Write boot ROM image to EBUSC1      r. Write ROM image  
4. Copy boot ROM image to EBUSC1      t. Clear flash ROM  
y. Simple access test                 i. Access test  
  
x. Exit                                z. Go to main menu
```

```
command>
```

8. Power the system off.
9. Set switch bank 2 as specified earlier to run OS-9.

The serial cable should now be connected to PJ19 (CH0) with a setting of 38,400 baud, 8 bit words, no parity, and 1 stop bit. Once you apply power, the OS-9 system should boot.

## Optional Procedures

The following sections detail procedures you may perform once you have installed and configured OS-9.

### 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 rom (coreboot + bootfile) image and burn the image into flash using `pflash`.

1. If you are no longer running the Configuration Wizard (from the previous steps) on your Windows desktop, select **Start -> Programs -> RadiSys -> Microware OS-9 for MIPS64 -> 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**.
2. Select **Configure -> Build Image..** to display the **Master Builder** screen.
3. Ensure **Coreboot + Bootfile** is selected and click **Build**.
4. Once the build is complete, click **Save As** to save the ROM image to a directory of your choosing.

5. The default location for the `rom` file is in the following directory:  
`\MWOS\OS9000\MIPS64\PORTS\RBHM4000\BOOTS\INSTALL\PORTBOOT`
6. Start a DOS shell on the host system.
7. Navigate to the directory in which the OS-9 ROM image, `rom`, is located.
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
```
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>
```
10. Log in with the username `super` and the password `user`.
11. At the `ftp>` prompt, enter the following command:  

```
ftp> cd /r1
```

This specifies that the `/r1` device's root is the current directory.
12. At the `ftp>` prompt, enter the following command:  

```
ftp> bin
```

This designates binary format.
13. At the `ftp>` prompt, enter the following command:  

```
ftp> put rom
```

The OS-9 ROM image file is transferred to the target's RAM disk (`/r1`).
14. On the target, enter the following command:  

```
$ pflash /r1/rom
```

The file is programmed into the target system's flash memory for the next reboot of the system.

### Flashed `rom` Image Issues

There are a number of issues to keep in mind when programming the flash with binary images:

If OS-9 fails to boot after using `pflash` to write the `coreboot` or `rom` image, you must revert back to using HCP5 to program the flash.

If a `rom` image (`coreboot + bootfile`) with an uncompressed bootfile is programmed into the flash, but an Ethernet or kermit boot is used, any modules not found in the downloaded boot will be found and used from the flash. The "`nokrs=1`" option can be used with the Ethernet or kermit booters (e.g. `eb nokrs=1` at the ROM boot menu) to prevent modules in flash from being found.

To erase the `bootfile` portion of a `rom` image programmed into flash, erase the entire part and then reprogram with only `coreboot`.

## Building with Makefiles

Building boots with makefiles allows you greater control over which modules are included in the boot. For the Toshiba RBHM4x00 reference board, the directory in which boots can be made is listed below:

```
MWOS/OS9000/MIPS64/PORTS/RBHM4000/BOOTS/SYSTEMS/RBHM4<x>00
```

### Makefile Network Option

Networking is not included in a bootfile by default. To include the networking modules in the bootfile, set the **NETWORK** macro definition to **TRUE** in the file named `makefile`. In addition, be certain that the IP address for the board is setup correctly; this helps to avoid network problems.

### Using Makefiles

When using a makefile to build boots, three bootlist files are used to include the modules for booting. These bootlist files can be edited to include or exclude modules required for the system.

These bootfile lists are located in `<PORTS>/BOOTS/SYSTEMS/RBHM4<x>00`, and are defined as follows:

<code>coreboot.m1</code>	used to make the low-level boot (called <code>coreboot</code> )  When using this file, the <code>romcore</code> file must be input first, followed by the <code>initext</code> file. These two files are not OS-9 modules. <code>romcore</code> is the raw code needed to bring the hardware to a known stable state, while <code>initext</code> is a way for users to extend the low level <code>sysinit</code> code without changing <code>sysinit.c</code> or remaking <code>romcore</code> .  The rest of the files included with <code>coreboot.m1</code> are actual OS-9 modules. Low-level booters and debuggers can be added or removed. In addition, the low-level Ethernet, IP stack, and SCSI system can be uncommented in order to provide <code>bootp</code> booting and/or SCSI booting. Low-level Ethernet or low-level SLIP can also provide system state debugging through Hawk.
<code>bootfile.m1</code>	used to create the high-level boot (called <code>bootfile</code> )  This file contains all of the modules needed to produce an OS-9 system. This includes the kernel, system protection, cache control, file managers, and drivers and descriptors. Also included are various utilities and application programs.  Not included with this file are networking modules. Additional modules can be included or excluded where appropriate.
<code>spf_mods.m1</code>	contains the SoftStax modules and network utilities

These modules are simply merged into the end of the bootfile created from the `bootfile.ml` bootlist.

### Making Network Configuration Changes

To configure the network parameters for SoftStax and Ethernet, one file needs to be altered and one makefile needs to be run. To do this, complete the following steps:

1. Navigate to the `MWOS\OS9000\MIPS64\PORTS\RBHM4000\SPF\ETC` directory and open the `interfaces.conf` file.
2. From the `interfaces.conf` file, fill in the correct IP address, broadcast address, and netmask values. You can also supply the host name in this area as well.
3. Save the file.

Once you have saved the file, run the makefile in the directory listed in step one. This makes the appropriate `inetdb` and `inetdb2` modules.



Because the Configuration Wizard configures the network in its own manner, if you are using it to configure network parameters, the above changes are not needed. However, if you choose to make the above changes, the Wizard will remain unaffected.

### Low-Level Network Configuration Changes

To configure the low-level Ethernet parameters, one file needs to be altered and one makefile needs to be run. To do this, complete the following steps:

1. Navigate to the `MWOS\OS9000\MIPS64\PORTS\RBHM4000\ROM\CNFGDATA` directory and open the `config.des` file.
2. From the `config.des` file, you will need to correctly define the macros for the IP address, broadcast, subnet, and mac.
3. Run the makefile in the directory listed in step one and a new `cnfgdata` module will be created. A coreboot can now be created with this configuration.



Because the Configuration Wizard configures the network in its own manner, if you are using it to configure Ethernet parameters, the above changes are not needed. However, if you choose to make the above changes, the Wizard will remain unaffected.



# 2

## Board Specific Reference

---



This chapter contains porting information specific to the Toshiba RBHM4x00 board. It includes the following sections:

[Boot Options](#)

[The Fastboot Enhancement](#)

[OS-9 Vector Mappings](#)

[Port Specific Utilities](#)

## 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\MIPS64\PORTS\RBHM4000\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\MIPS64\PORTS\RBHM4000\ROM\ROMCORE
```

<code>bo</code>	ROM boot: the system runs "in-place", from the flash. This makes it impossible to set a breakpoint or trace with the debugger since the code in flash cannot be modified.
<code>lr</code>	load to RAM boot: the system copies the ROM bootfile image into RAM and runs from there. Thus, breakpoints and tracing will work as expected.

## Booting over a Serial Port via kermi

The system can download a bootfile in binary form over its serial port at speeds up to 38400 using the kermi protocol. Dots on the console indicate download progress.

<code>ker</code>	kermi boot: The boot file is sent via kermi protocol into system RAM and it runs from there.
------------------	--

## Restart Booter

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

<code>q</code>	quit: Quit and reboot the board 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 Toshiba RBHM4400 example boot using the `bo` boot option.

```
OS-9 Bootstrap for MIPS (Edition 68)
```

```
PCI device initialization - Completed
```

```
LLPRO100: Intel PCI EtherExpress Pro100 - PCI Device ID 0x1229
```

```
LLPRO100: PCI device located @ BUS:DEV [0000:000C]
```

```
LLPRO100: IP Address [010.020.003.205] MAC Address
```

```
[00:02:b3:09:e4:ce]
```

```
LLPRO100: Default Gateway 0x0A140201 Subnet Mask 0xFFFFFE00
```

```
LLPRO100: PCI I/O address 0xAFF00000
```

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

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

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

```
Boot over Ethernet (INTEL PRO100) -- <eb>
```

```
Boot embedded OS-9 in-place ----- <bo>
```

```
Copy embedded OS-9 to RAM and boot - <lr>
```

```
Kermit download ----- <ker>
```

```
Enter system debugger ----- <break>
```

```
Restart the System ----- <q>
```

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

```
Now searching memory ($9fc55000 - $9fffffff) for an OS-9 Kernel...
```

```
An OS-9 kernel was found at $9fc55000
```

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

```
+3
```

```
+5
```

```
$ mfree
```

```
Current total free RAM: 52824.00 K-bytes
```

## The Fastboot Enhancement

The Fastboot enhancements to OS-9 were added to address the needs of embedded systems that require faster system bootstrap performance. The Fastboot concept exists to inform OS-9 that the defined configuration is static and valid. This eliminates the dynamic search OS-9 usually performs during the bootstrap process. It also allows the system to perform for a minimal amount of runtime configuration. As a result, a significant increase in bootstrap speed is achieved.

### Overview

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

One important feature of the Fastboot enhancement is the ability of the flags to become dynamically altered during the bootstrap process. For example, the bootstrap code might be configured to query dip switch settings, respond to device interrupts, or respond to the presence of specific resources that indicate different bootstrap requirements.

Another important feature of the Fastboot enhancement is its versatility. The enhancement's versatility allows for special considerations under a variety of circumstances. This can be useful in a system in which most resources are known, static, and functional, but whose additional validation is required during bootstrap for a particular instance (such as a resource failure).

### Implementation Overview

The Fastboot configuration flags have been implemented as a set of bit fields. One 32-bit field has been dedicated for bootstrap configuration. This four-byte field is contained within a set of data structures shared by the kernel and the ModRom sub-components. Hence, the field is available for modification and inspection by the entire set of system modules (both high-level and low-level).

Currently, there are six-bit flags defined, with eight bits reserved for user-definable bootstrap functionality. The reserved user-definable bits are the high-order eight bits (31-24). This leaves bits available for future enhancements. The currently defined bits and their associated bootstrap functionality are listed in the following sections.

#### **B\_QUICKVAL**

The `B_QUICKVAL` bit indicates that only the module headers of modules in ROM are to be validated during the memory module search phase. Limiting validation in this manner will omit the CRC check on modules, which may save you a considerable amount of time. For example, if a system has many modules in ROM, in which access time is typically longer than it is in RAM, omitting the CRC check will drastically decrease the bootstrap time. Furthermore, since it is rare that data corruption will occur in ROM, omitting the CRC check is a safe option.

In addition, the `B_OKRAM` bit instructs the low-level and high-level systems to accept their respective RAM definitions without verification. Normally, the system probes

memory during bootstrap based on the defined RAM parameters. This method allows system designers to specify a possible range of RAM the system will validate upon startup; thus, the system can accommodate varying amounts of RAM. However, in an embedded system (where the RAM limits are usually statically defined and presumed to be functional) there is no need to validate the defined RAM list. Bootstrap time is saved by assuming that the RAM definition is accurate.

### **B\_OKROM**

The `B_OKROM` bit causes acceptance of the ROM definition without probing for ROM. This configuration option behaves similarly to the `B_OKRAM` option with the exception that it applies to the acceptance of the ROM definition.

### **B\_1STINIT**

The `B_1STINIT` bit causes acceptance of the first `init` module found during cold-start. By default, the kernel searches the entire ROM list passed up by the ModRom for `init` modules before it takes the `init` module with the highest revision number. Using the `B_1STINIT` in a statically defined system omits the extended `init` module search, which can save a considerable amount of time.

### **B\_NOIRQMASK**

The `B_NOIRQMASK` bit instructs the entire bootstrap system to not mask interrupts for the duration of the bootstrap process. Normally, the ModRom code and the kernel cold-start mask interrupts for the duration of the system startup. However, in systems with a well-defined interrupt system (systems that are calmed by the `sysinit` hardware initialization code) and a requirement to respond to an installed interrupt handler during startup, this option can be used. Its implementation will prevent the ModRom and kernel cold-start from disabling interrupts. (This is 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 has not been omitted, the `B_NOPARITY` bit causes the system to not perform parity initialization of the RAM. Parity initialization occurs during the RAM probe phase. The `B_NOPARITY` option is useful for systems that either require no parity initialization or only require it for “power-on” reset conditions. Systems that only require parity initialization for initial power-on reset conditions can dynamically use this option to prevent parity initialization for subsequent “non-power-on” reset conditions.

## **Implementation Details**

This section describes the compile-time and runtime methods by which you can control the bootstrap speed of your system.

## Compile-time Configuration

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

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

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

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

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

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

## Runtime Configuration

The default bootstrap configuration can be overridden at runtime by changing the `rinf->os->boot_config` variable from either a low-level P2 module or from the `sysinit2()` function of the `sysinit.c` file. The runtime code can query jumper or other hardware settings to determine which user-defined bootstrap procedure should be used. 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 have been performed. This means that any runtime override of the default settings pertaining to the memory search must be done from the code in the P2 module code.

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

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

## OS-9 Vector Mappings

This section contains the OS-9 vector mappings for the RBHM4x00 Embedded TX49 boards.

The MIPS64 standard defines exceptions 0x0-0x1f. The OS-9 system maps these one-to-one. External interrupts from vector 0x0 expand to the virtual vector range shown below by the `vect49xx` module.

**Table 2-1. OS-9 IRQ Assignment for All MIPS64 Processors**

OS-9 IRQ #	MIPS64 Vector Description
0x0	Internal/External Interrupt
0x1	TLB Modification Exception
0x2	TLB Load Exception
0x3	TLB Store Exception
0x4	Address Error Exception (Read)
0x5	Address Error Exception (Write)
0x6	BUS Error Exception (Fetch)
0x7	BUS Error Exception (Read/Write)
0x8	Syscall Exception
0x9	Breakpoint Exception
0xa	Reserved Instruction Exception
0xb	Co-Processor Unusable Exception
0xc	Arithmetic Overflow Exception
0xd	Trap Exception
0xf	Floating-point Exception
0x12	Precise Co-processor 2 Exception
0x16	MDMX Unusable Exception
0x17	Watchpoint Exception
0x18	Machine Check Exception
0x1e	Cache Error

**Table 2-2. Toshiba RBHM4x00 Specific IRQ Assignments**

OS-9 IRQ #	RBHM4x00 Specific Vector Description
0x20	User-state TLB Miss Exception
0x50	ECC Error
0x51	TX49 Write Timeout Error
0x52	INT[0] - Serial I/O DSR0
0x53	INT[1] - I/O Controller (FPGA) -- Serial I/O DSR0, PCI INTA/B/C/D
0x54	INT[2] - PCI INTB
0x55	INT[3] - On-board 10MB Ethernet
0x56	INT[4] - PCI INTA

Table 2-2. Toshiba RBHM4x00 Specific IRQ Assignments (Continued)

OS-9 IRQ #	RBHM4x00 Specific Vector Description
0x57	INT[5] - PCI INTB
0x58	Serial I/O 0
0x59	Serial I/O 1
0x5a	DMA0[0]
0x5b	DMA0[1]
0x5c	DMA0[2]
0x5d	DMA0[3]
0x5e	Programmable Interrupt Controller
0x5f	PCI DMA Controller
0x60	PCI Controller
0x61	Timer 0
0x62	Timer 1
0x63	Timer 2
0x64	Reserved
0x65	Reserved
0x66	PCI ERR
0x67	PCI PMC
0x68	Reserved
0x69	Reserved
0x6a	Reserved
0x6b	DMA1[0]
0x6c	DMA1[1]
0x6d	DMA1[2]
0x6e	DMA1[3]
0x6f	Reserved

## Port Specific Utilities

Utilities for the Toshiba RBHM4x00 boards are located in the following directory:

MWOS/OS9000/MIPS64/PORTS/RBHM4000/CMD5

The following port specific utilities are included:

<a href="#">dmppci</a>	Shows PCI device information.
<a href="#">pciv</a>	Displays board PCI bus information.
<a href="#">pflash</a>	Programs On-board flash.
<a href="#">setpci</a>	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.

The following is an example display of an Intel Ethernet Pro 10/100 PCI board:

```
$ dmppci 0 12 0
```

```
PCI DUMP Bus:0 Dev:12 Func:0 Size:64
```

```
-----
VID  DID  CMD  STAT CLASS  RV CS  IL  IP  LT  HT  BI  MG  ML  SVID  SDID
---  ---  ---  ---  ---  ---  ---  ---  ---  ---  ---  ---  ---  ---  ---
8086 1229 0007 0290 020000 08 08 53 01 20 00 00 08 38 8086 000c

BASE[0]  BASE[1]  BASE[2]  BASE[3]  BASE[4]  BASE[5]  CIS_P  EXROM
-----  -----  -----  -----  -----  -----  -----  -----
---
08100000 0ff00001 08000000 00000000 00000000 00000000 00000000 00000000
00000000

Offset 00 01 02 03 04 05 06 07 08 09 0a 0b 0c 0d 0e 0f
-----  -----  -----  -----  -----  -----  -----  -----  -----  -----  -----  -----  -----  -----  -----  -----
0000    86 80 29 12 07 00 90 02 08 00 00 02 08 20 00 00
0010    00 00 10 08 01 00 f0 0f 00 00 00 08 00 00 00 00
0020    00 00 00 00 00 00 00 00 00 00 00 00 86 80 0c 00
0030    00 00 00 00 dc 00 00 00 00 00 00 00 53 01 08 38
```

**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.

**Description**

The `pciv` utility allows visual indication of the status of the PCI bus.

The following is an example display:

```
$ pciv -a
BUS:DV:FU VID DID CMD STAT CLASS RV CS IL IP
-----
000:12:00 8086 1229 0007 0290 020000 08 08 53 01
(C) [32-bit] base_addr[0] = 0x08100000 PCI/MEM 0xa8100000 Size = 0x00001000
(C) [32-bit] base_addr[1] = 0x0ff00001 PCI/IO 0xaff00000 Size = 0x00000040
(C) [32-bit] base_addr[2] = 0x08000000 PCI/MEM 0xa8000000 Size = 0x00100000
Network Controller [S]
```

## pflash

### Program Toshiba Flash

#### Syntax

```
pflash { [options] } <filename>
```

#### Options

-b [=] addr	Specify base address of flash (hex). The default is 0xbfc00000 for RBHM4200 and 0xbf000000 for RBHM4400.
-eu	Erase used space only. This is the default mode of operation.
-ew	Erase all of flash. This will erase all TrueFFS and/or OS-9 boot code stored in the flash.
-i	Print information about flash.
-ne	Do not erase flash.
-nv	Do not verify erase or write operations.
-q	Do not display progress messages and spinning indicator.
-s [=] addr	Specify write address for <filename> in hexadecimal. The default start address is 0xbfc00000.
-u	Leave flash part unlocked upon completion.
-z [[=] <file>]	Read additional command line arguments from <file>. The default <file> is standard input.

#### Description

The `pflash` utility allows programming of the Toshiba flash parts. The primary use is in burning the OS-9 ROM or coreboot image into the on-board flash parts. This allows for booting using the lr/bo booters.

`pflash` requires <filename>, `-i`, and/or `-ew`.

<filename>'s size need not be a multiple on the flash's erase block size. `pflash` will only reprogram the part of the flash that <filename> will occupy. The remainder of the flash remains undisturbed. This feature allows for small updates within flash erase blocks.

For this board, `pflash` does no locking nor unlocking of the flash parts. Thus, `-u` has no effect.

## setpci

### Set PCI Value

#### Syntax

```
setpci <bus> <dev> <func> <offset> <size> {<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 (e.g. command register offset = 4).
<size>	Size b = byte, w = 16-bit word, or d = 32-bit double word.
<value>	An optional value to write. If no <value> is specified, <code>setpci</code> will read and display the value at the specified offset.



# A

## Board Specific Modules

---



This chapter describes the modules specifically written for the Toshiba RBHM4x00 board. It includes the following sections:

[Port-Specific Low-Level System Modules](#)

[Common Low-Level System Modules](#)

[Port-Specific High-Level System Modules](#)

[Port-Specific High-Level Utilities](#)

[Common High-Level System Modules](#)

## Port-Specific Low-Level System Modules

The following low-level system modules are tailored specifically for the Toshiba RBHM4x00 board. They are located in the following directory:

MWOS/OS9000/MIPS64/PORTS/RBHM4000/CMD5/BOOTOBJS/ROM

<code>cnfgdata</code>	contains low-level configuration data
<code>cnfgfunc</code>	provides access services to the <code>cnfgdata</code>
<code>commcnfg</code>	inits communication port defined in <code>cnfgdata</code>
<code>conscnfg</code>	inits console port defined in <code>cnfgdata</code>
<code>initext</code>	user-customizable system initialization module
<code>io4900</code>	low-level serial driver for serial ports
<code>llne2000</code>	low-level Ethernet driver module
<code>llpro100</code>	low-level Ethernet driver module
<code>pciwalk</code>	inits devices found on the PCI bus
<code>portmenu</code>	inits booters defined in the <code>cnfgdata</code>
<code>romcore_4200</code>	bootstrap code for the RBHM4200
<code>romcore_4400</code>	bootstrap code for the RBHM4400
<code>rpciv</code>	low-level booter used to display PCI bus information
<code>tmr4900</code>	low-level timer module for TX49 processors
<code>usedebug</code>	debugger configuration module

## Common Low-Level System Modules

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

MWOS/OS9000/MIPS64/CMD5/BOOTOBJS/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>excption</code>	provides low-level exception services
<code>flshcach</code>	provides low-level cache management services
<code>hlproto</code>	provides user level code access to protoman
<code>llbootp</code>	provides bootp services
<code>llip</code>	provides low-level IP services
<code>llkermit</code>	provides a booter that uses kermit protocol

<code>llslip</code>	provides low-level SLIP services
<code>lltcp</code>	provides low-level TCP services
<code>lludp</code>	provides low-level UDP services
<code>notify</code>	provides state change information for use with LL and HL drivers
<code>override</code>	provides a booter that allows a choice between menu and auto booters
<code>parser</code>	provides argument parsing services
<code>protoman</code>	provides a protocol management module
<code>restart</code>	provides a booter that causes a soft reboot of the system
<code>romboot</code>	provides a booter that allows booting from ROM
<code>rombreak</code>	provides a booter that calls the installed debugger
<code>rombug</code>	provides a low-level system debugger
<code>sndp</code>	provides low-level system debug protocol
<code>srecord</code>	provides a booter that accepts S-Records
<code>swtimer</code>	provides timer services via software loops

## Port-Specific High-Level System Modules

The following OS-9 system modules are tailored specifically for the Toshiba RBHM4x00 board. Unless otherwise specified, each module is located in the following directory:

`MWOS/OS9000/MIPS64/PORTS/RBHM4000/CMD5/BOOTOBS`

<code>INIT/nodisk</code>	system configuration module for a diskless OS-9 boot
<code>picsub</code>	programmable interrupt controller handling module
<code>tk3927</code>	system ticker module
<code>hcsb</code>	provides a high speed timer interface used by the HawkEye profiler (not present if HawkEye is not installed)
<code>rtc1742</code>	battery backed real-time clock module
<code>sc3927</code>	serial port driver. The descriptors for this driver are found in the <code>DESC/SC3927</code> sub-directory.

### Descriptors `/term1 /t1`

`term1` and `t1` are assigned to SIO0. The connector for SIO0 is labeled as PJ19/CH0.

Driver Name: `sc3927`

Default Baud Rate: 38,400

Default Parity: None

Default Data Bits: 8

Default Stop Bits: 1

To use it: Select **SIO0** in the Configuration Wizard.

### Descriptors /term2 /t2

term2 and t2 are assigned to the SIO1. The connector for SIO1 is labeled as PJ20/CH1.

Driver Name: `sc3927`

Default Baud Rate 38,400

Default Parity: None

Default Data Bits: 8

Default Stop Bits: 1

To use it: Select **SIO1** in the Configuration Wizard.

### Baud Rates

The following baud rates are supported by the `sc3927` driver: 75, 110, 150, 300, 600, 1200, 1800, 2000, 2400, 3600, 4800, 7200, 9600, 19200, 31250 and 38400.

`vect49xx`

exception and interrupt handling code for TX49 series processors.

## Port-Specific High-Level Utilities

The following RBHM4x00-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/MIPS64/PORTS/RBHM4000/CMDS`

<code>dmppci</code>	Allows a specific PCI device's configuration area to display.
<code>pciv</code>	Displays configuration information about all available PCI devices.
<code>pflash_4200</code>	Programs the on-board Toshiba flash devices on the RBHM4200 board. The module name in this file is <code>pflash</code> .
<code>pflash_4400</code>	Programs the on-board Toshiba flash devices on the RBHM4400 board. The module name in this file is <code>pflash</code> .
<code>setpci</code>	Allows changes to the PCI configuration of devices.

## Common High-Level System Modules

These files are located in the following directory:

`MWOS/OS9000/MIPS64/CMDS/BOOTOBS`

<code>kernel</code>	Provides all basic services for the OS-9 system.
<code>cach49xx</code>	Provides cache control for the CPU cache hardware. The

	module name in this file is <code>cache</code> .
<code>fpu64</code>	Provides handling of the floating-point hardware. The module name in this file is <code>fpu</code> .
<code>ioman</code>	Provides common I/O support for the operating system.
<code>mq</code>	Device descriptor for inter-process message queues.
<code>msgman</code>	File manager that provides support for inter-process message queues.
<code>pcf</code>	Random block device management functions for MS-DOS FAT format.
<code>pipeman</code>	Memory FIFO buffer management for inter-process communication.
<code>rbf</code>	Generic random block device management functions for the OS-9 format.
<code>scf</code>	Sequential character device management functions.
<code>spf</code>	Generic protocol device management function support.
<code>ssm64</code>	System Security Module—provides support for the CPU's MMU (Memory Management Unit). The module name in this file is <code>ssm</code> .

