OS-9 for Hitachi HS8001 Board Guide

Version 3.2

www.radisys.com
World Headquarters
5445 NE Dawson Creek Drive • Hillsboro, OR
97124 USA
Phone: 503-615-1100 • Fax: 503-615-1121
Toll-Free: 800-950-0044
International Headquarters
Gebouw Flevopoort • Televisieweg 1A
NL-1322 AC • Almere, The Netherlands
Phone: 31 36 5365595 • Fax: 31 36 5365620
RadiSys Microware Communications Software Division, Inc.
1500 N.W. 118th Street
Des Moines, Iowa 50325
515-223-8000

Revision A
February 2002

Copyright and publication information

This manual reflects version 3.2 of Enhanced OS-9 for SH5.
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.

Table of Contents

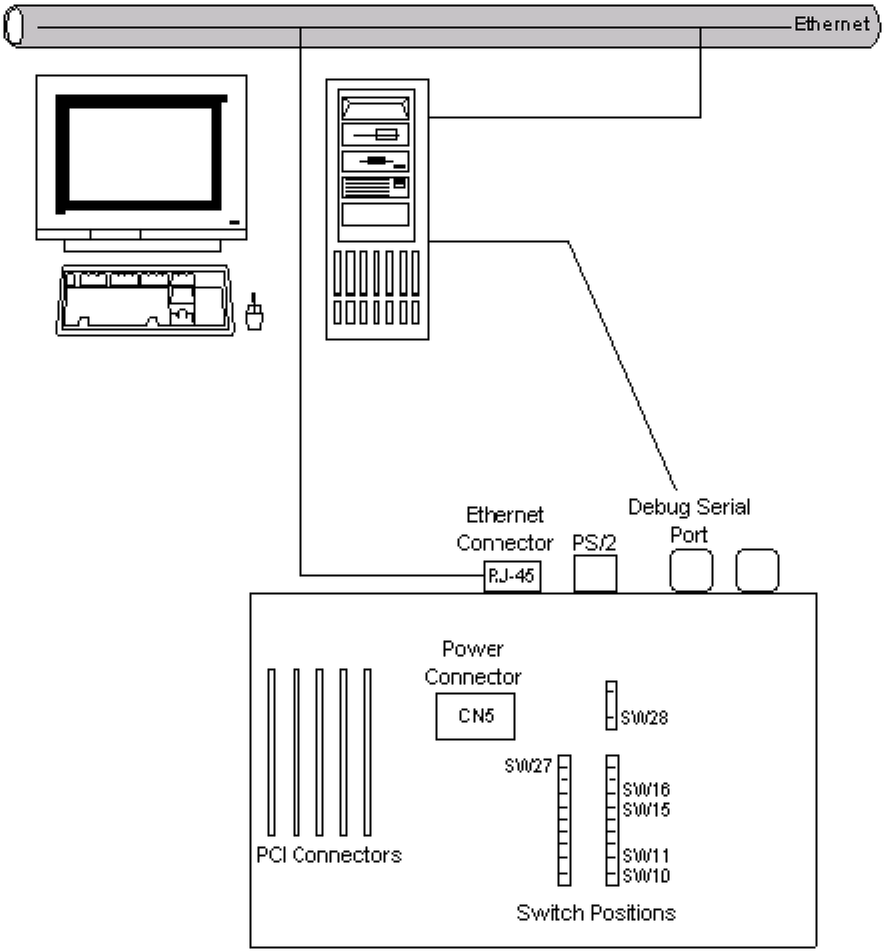
Chapter 1: Installing and Configuring OS-9

This chapter describes installing and configuring OS-9® on the Hitachi HS8001 board. It includes the following sections:

Development Environment Overview

Figure 1-1 shows a typical development environment for the Hitachi HS8001 board. The following illustration shows the minimum equipment required for software development using OS-9 on the HS8001 board.

Figure 1-1 Hitachi HS8001 Development Environment



Requirements and Compatibility

Note

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

Host Hardware Requirements (PC Compatible)

Your host PC must have the following minimum hardware characteristics:

- 32MB of RAM
- an Ethernet network card

Host Software Requirements (PC Compatible)

Your host PC must have the following software installed:

- Windows 95, 98, ME, 2000, or NT

Target Hardware Requirements

Your MIPS evaluation board requires the following hardware:

- a power supply
- an RS-232 null modem serial cable (for serial console)
- an Ethernet cable or a second RS-232 null modem serial cable (for downloading programs to the board)

Connecting the Target to the Host

Connecting the Hitachi HS8001 to your host PC involves attaching cables to the reference board, configuring switch settings for the Hitachi Debug Monitor, and booting the target. Once you have done these tasks, you can use the serial console in Hawk to verify the serial connection.

Attaching the Cables

Complete the following steps to attach the cables to the reference board:

Step 1. Attach the power cable to the connector on the board (CN5).

Step 2. Connect a Null Modem cable to the Debug Serial Port. This serial port is the one closest to the PS/2 and RJ-45 Ethernet connector.

Step 3. Connect the other end of the Null Modem cable to COM1 on your host machine.

Configuring Switches for the Hitachi Debug Monitor

Before powering on the target, you will need to configure switches for the Hitachi Debug Monitor. The Hitachi Debug Monitor is the program that will be used to download the OS-9 image into RAM.

In order to use the Debug Monitor as required, you will need to configure the switches for use with OS-9. Most of the factory switch settings for the Hitachi Debug Monitor are appropriate for configuring OS-9. However, you will need to verify that the following switches are set as indicated:

Note

The switches listed below are labeled both in **Figure 1-1** and on your HS8001 board.

- Set SW10 and SW11 to 1.
- Set SW 27 and SW 28 to 0.
- Set SW15 and SW16 to 0.

For More Information

For more information regarding switch settings for Hitachi HS8001 board, refer to the ***CPU Board User's Guide***, included with your hardware.

Booting to the Hitachi Debug Monitor

Once you have attached the appropriate cables and configured the switch settings for the Hitachi Debug Monitor, you can safely power on the HS8001 board and boot to the Debug Monitor. To do this, complete the following steps:

Step 1. From your host machine, access and open your Hyperterminal program.

Step 2. Once Hyperterminal is open, establish a new connection by selecting `New Connection` from the **File** menu.

Step 3. Select a name for your connection and click `OK`.

Step 4. From the **Connet To** dialog, select the appropriate COM port for your connection and click `OK`.

Step 5. Under the **Port Settings** dialog, apply the followng settings:

- Bits per second: `115200`
- Data Bits: `8`
- Parity: `None`
- Stop bits: `1`
- Flow Control: `Hardware`

Click `OK`.

Step 6. Apply power to the HS8001 board. You should now see a screen appear on your host PC for the Hitachi Debug Monitor. It should look similar to the following example:

```
SH8000 CPU BOARD Debug Monitor V0.96-B 010702
Boot from DDR Memory           WS1.2 Release
Copyright (C) Hitachi, LTD. 2001
Licensed Material of Hitachi, Ltd.
:
```

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 (or bootfile) image as specified by its configuration. For example from a FLASH part or Ethernet network. It is also responsible for building basic structures based on the image it finds and passing control to the kernel to bring up the OS-9 system.

Bootfile

The bootfile image contains the kernel and other high-level modules (initialization module, file managers, drivers, descriptors, 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.

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

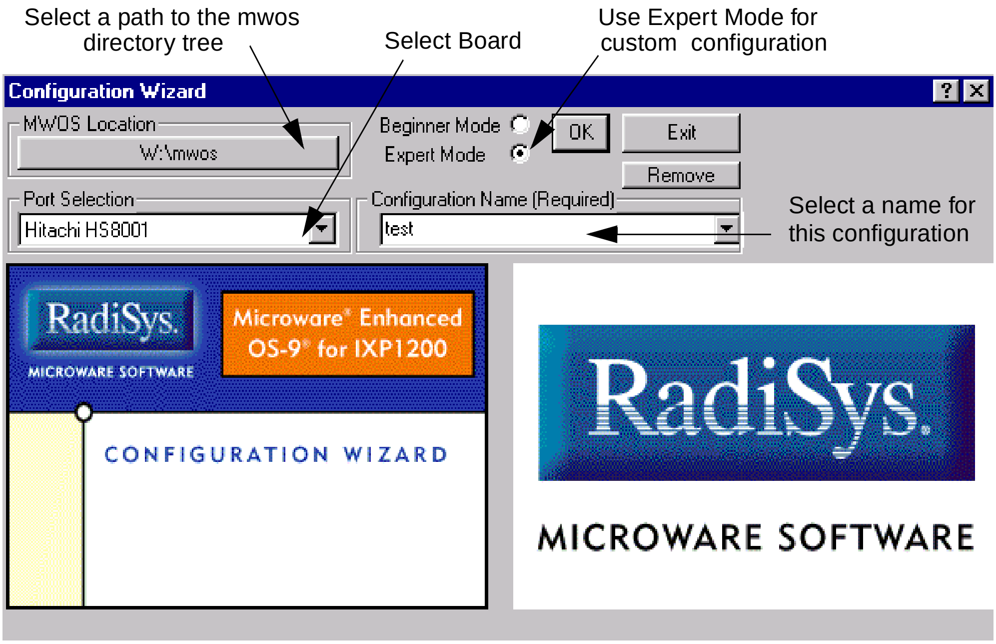
Using the Configuration Wizard

This section describes how to use the Configuration Wizard to build the OS-9 ROM image. To open and use the Wizard, complete the following steps:

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

Step 2. On the Windows desktop, select `Start` --> `Programs` --> `Microware` --> `Enhanced OS-9 for <processor>` --> `Microware Configuration Wizard`. You should see the following opening screen.

Figure 1-2 Configuration Wizard Opening Screen



Step 3. Verify that the MWOS location is correct. If not, click on the MWOS button to select the path to the MWOS directory structure's location.

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

Step 5. Select a name for your configuration in the **Configuration Name** field. This enables you to modify and save your settings for the ROM image incrementally, without having to reselect every option for each change.

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

For More Information

The ***OS-9 Device Descriptor and Configuration Module Reference*** manual included on your CD describes each of the OS-9 modules and the various ways that the software can be configured to meet your needs.

Configuring Coreboot Options

Most of the default options for the configuration of the coreboot are correct. There are a few functions, however, such as Ethernet, that need additional information in order to be configured correctly. To configure the coreboot options as necessary, complete the following steps:

Step 1. From the menu bar, select `Configure` -> `Coreboot` -> `Main Configuration`.

Step 2. Click on the **Debugger** tab. Make sure **Ethernet** is selected in the **Remote Debug Connection** area and **Remote** is selected in the **Select Debugger** area. Remote debugging is enabled so that system-state debugging can be performed in Hawk.

Step 3. Click the **Ethernet** tab and enter the Ethernet address information in the address text boxes. For most situations you will need to fill out the following information:

- IP address
- IP broadcast
- IP gateway
- subnet mask

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

Step 4. Click `OK` to close the window.

Configuring Bootfile Options

Most of the default options for the configuration of the bootfile are correct. There are a few functions, however, such as Ethernet, that need additional information in order to be configured correctly. To configure the bootfile options as necessary, complete the following steps:

Step 1. To configure the Ethernet function, select `Configure` -> `Bootfile` -> `Network Configuration` from the menu.

Step 2. Click on the **Interface** tab.

Step 3. Click `Ethernet Connection` in the **Select Interface** list.

Step 4. Make sure that **Specify an IP Address** is selected and the address information in the IP address text boxes is correct. Each should have been copied from the coreboot Ethernet dialog box.

Step 5. Select the **Ethernet** check box in the **Enable Interface** area.

Step 6. Make sure the name of the Ethernet controller chip is displayed in the combo box located to the right of the **Enable Interface** area. If it is not visible, the Ethernet modules will not be included in the build.

Step 7. Click on the **SoftStax Setup** tab, and select `Enable SoftStax`.

Step 8. Click `OK` to close the dialog box.

Step 9. Select `Configure` -> `Bootfile` -> `Disk Configuration` from the menu and verify that the default settings in the dialog box are acceptable.

Step 10. Select `Configure` -> `Build Image` from the menu to display the **Master Builder** window.

Step 11. Select the following check boxes as appropriate:

- SoftStax (SPF) Support
- User State Debugging Modules

In addition, if you are using a RAM disk, select `Disk Support` and `Disk Utilities`.

Step 12. Click `Coreboot + Bootfile` and then click `Build`. This builds the ROM image that can be burned into flash memory. The name of the file containing the ROM image is `rom.S`. It is in the Motorola S-record format. The file `rom.s` is located in the following directory:

```
mwos\OS9000\MIPS32\PORTS\IDT_79EB355\BOOTS\INSTALL\PORTBOOT
```

(The directory above should have been specified as the outgoing directory when the TFTP server was intially set up.)

Step 13. Click `Finish` and then select `File` -> `Save Settings` to save the configuration.

Step 14. Select `File` -> `Exit` to exit the Configuration Wizard.

Transferring the ROM Image to the Target

The Hitachi Debug Monitor will be used to transfer the OS-9 ROM image you created in the **Building the OS-9 ROM Image** section to the HS8001 board and program it into flash. Complete the following steps to transfer the image to the board:

Step 1. From the Hitachi Debug Monitor, type the `ld` command. The Monitor program should echo the following text:

```
Please load from host.
```

Step 2. From HyperTerminal, select `Transfer -> Send Text File`. Browse to the `rom.S` file and select `OK`. The file should now download to the HS8001 board.

Step 3. After the ROM file has completed downloading, enter the following commands in the Hitachi Debug Monitor to program it into flash:

```
pf 80000000 0
pf 80040000 1
pf 80080000 2
pf 800c0000 3
pf 80100000 4
pf 80140000 5
pf 80180000 6
pf 801c0000 7
```

Each command will program 256K of flash. Thus, the preceeding commands will program a ROM image that is 2MB.

Setting the Switches to Boot OS-9

The following switches will need to be changed to boot OS-9 directly:

Note
Be sure to power off before adjusting these switches.

- SW 10 and SW 11 should be turned to 1.
- SW 28 should be turned to 0.

Note
The switches listed above are labeled both in **Figure 1-1** and on your HS8001 board.

Optional Procedures

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

Using OS-9 to Program the Flash

After you have successfully booted the board with OS-9, you can use the `pflash` command to program the flash with an updated bootfile. Below is a list of steps that will show you how to download a file to the HS8001 board running OS-9, and how to use the `pflash` command to program the bootfile into flash.

Step 1. Type `chd /r1` on the command line for the target. This command initiates a large RAM disk that will hold the bootfile.

Step 2. Use FTP to download the bootfile to the `/r1` ram disk.

Step 3. Type `pflash -f=bootfile -un` to program the flash.

Note

If you are programming a ROM image into flash instead of a bootfile, use the following command inplace of `pflash -f=bootfile -un`.

```
pflash –f=rom –un –ri
```

Chapter 2: Board Specific Reference

This chapter contains porting information specific to the HS8001 board. It includes the following sections:

- **The Fastboot Enhancement**
- **OS-9 Vector Mappings**

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 eliminate 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 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.)

Note

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

B_NOPARITY

If the RAM probing operation has not been omitted, the B_NOPARITY bit causes the system to not perform parity initialization of the RAM. Parity initialization occurs during the RAM probe phase. The B_NOPARITY option is useful for systems that either require no parity initialization 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 overriding value of the macro should be established as a redefinition of the macro in the rom_cnfg.h header file or a macro definition parameter in the compilation command.

The rom_cnfg.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_cnfg.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.

Note

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

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

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

OS-9 Vector Mappings

Table 2-1 shows the OS9 IRQ assignment for the target board.

Table 2-1 IRQ Assignments

OS9 IRQ #	Hitachi HS8001 Function
0x92	Processor Interrupt Line (SuperIO/Lan interrupts)
0x95	Processor Interrupt Line (External Interrupts)
0x98	Processor Interrupt Line 2 (PCI bus 1 interrupts)
0x9a	Processor Interrupt Line 3 (PCI bus 2 interrupts)
0xb2	DMA Channel 0
0xb3	DMA Channel 1
0xb4	DMA Channel 3
0xb5	DMA Error
0xa0	Timer 0
0xa1	Timer 1
0xa2	Timer 2
0xa3	TMU
0xa4	T_RTC_ATI
0xa5	T_RTC_PRI
0xa6	T_RTC_CUI

Table 2-1 IRQ Assignments (continued)

OS9 IRQ #	Hitachi HS8001 Function
0xb8	Debug Serial Error
0xb9	Debug Serial Receive
0xba	Debug Serial BRI
0xbb	Debug Serial Transmit
0xab	Watchdog Timer
0xe0	Debug Serial Port
0xe1	Ethernet
0xe2	SuperIO Keyboard
0xe3	SuperIO Serial2
0xe4	SuperIO Serial1
0xe5	SuperIO Parallel
0xe6	SuperIO Mouse
0xe7	SuperIO IDE
f0-ff	External Interrupts
0xc0	PCI Bus1—INTA
0xc1	PCI Bus1—INTB
0xc2	PCI Bus1—INTC

Table 2-1 IRQ Assignments (continued)

OS9 IRQ #	Hitachi HS8001 Function
0xc3	PCI Bus1—INTD
0xc8	PCI BUS2—INTA
0xc9	PCI BUS2—INTB
0xca	PCI BUS2—INTC
0xcb	PCI BUS2—INTD
0xcc	PCI BUS2—FAL
0xcd	PCI BUS2—DEG
0xce	PCI BUS2—INTP
0xcf	PCI BUS2—INTS

Appendix A: Board Specific Modules

This chapter describes the modules specifically written for the Hitachi HS8001 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 HS8001 board. They are located in the following directory:

`MWOS/OS9000/SH5M/PORTS/HS8001/CMDS/BOOTOBJS/ROM`

`cnfgdata`	contains low-level configuration data
`cnfgfunc`	provides access services to the `cnfgdata`
`commcnfg`	inits communication port defined in `cnfgdata`
`conscnfg`	inits console port defined in `cnfgdata`
`initext`	user-customizable system initialization module
`io16550`	ROM based serial I/O driver
`io1scif`	ROM based serial I/O driver
`lle91c100`	Low-level Ethernet ROM driver
`portmenu`	inits booters defined in the `cnfgdata`
`romcore`	bootstrap code
`tmr8001`	ROM timer services
`usedebug`	debugger configuration module
`ide`	ROM based IDE driver
`pciwalk`	pci initialization code

High-Level System Modules

The following OS-9 system modules are tailored specifically for the SH-5 HS8001 boards. Unless otherwise specified, each module is located in the following directory:

`MWOS/OS9000/Sh5M/PORTS/HS8001/CMDS/BOOTOBJS`

`sc16550`	Serial driver for the 16550 UART
`sc8001`	Serial driver for the SCIF UART
`sp91c100`	Ethernet driver module
`tk8001`	System clock module
`irq8001`	Interrupt handler module
`rb1003`	RBF IDE driver
`rtc8001`	Real-time clock driver
`sc8042k`	PS/2 mouse and keyboard driver
`gx_cl543`	Graphics driver for GXCL543 PCI card
`mp_msptr`	Mouse PS/2 protocol module for MAUI
`mp_xtkb`	Keyboard PS/2 protocol module for MAUI

Common System Modules List

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

`MWOS/OS9000/SH5M/CMDS/BOOTOBJS/ROM`

`bootsys`	provides booter registration services
`console`	provides console services
`dbgentry`	inits debugger entry point for system use
`dbgserv`	provides debugger services
`excption`	provides low-level exception services
`flshcach`	provides low-level cache management services
`hlproto`	provides user level code access to protoman
`llbootp`	provides `bootp` services
`llip`	provides low-level IP services
`llkermit`	provides a booter that uses kermit protocol
`llslip`	provides low-level SLIP services
`lltcp`	provides low-level TCP services
`lludp`	provides low-level UDP services
`notify`	provides state change information for use with LL and HL drivers
`override`	provides a booter that allows a choice between menu and auto booters

`parser`	provides argument parsing services
`protoman`	provides a protocol management module
fdman	OS-9 RBF file system ROM based service