Home

OS-9® for SuperH 7750SE01 Board Guide

Version 4.7

RadiSys
THE POWER OF WE

www.radisys.com

Revision A • July 2006

Copyright and publication information

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

Contents

1 Installing and Configuring OS-9®

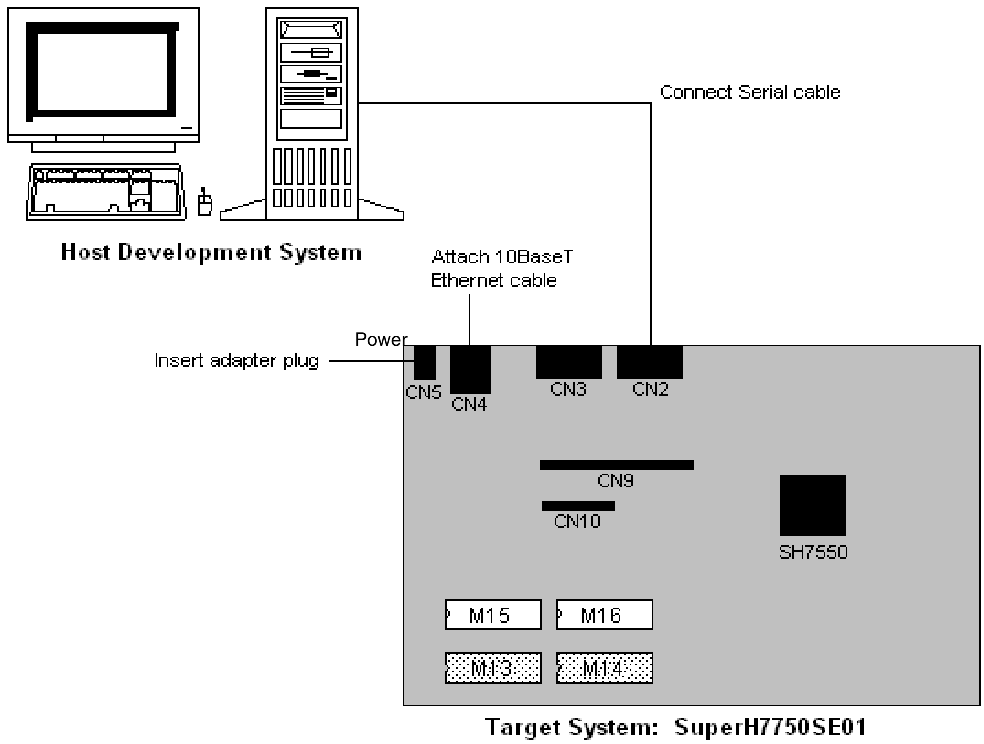
The procedures in this chapter are designed to help you to set up the SuperH evaluation board and create an bootfile image. The following sections are included:

- Development Environment Overview
- Requirements and Compatibility
- Target Hardware Setup
- Connecting the Target to the Host
- Building the OS-9 Bootfile Image
- Creating a Startup File

Development Environment Overview

Figure 1-1 shows a typical development environment for the SuperH 7750SE01 board. These components include the minimum required to enable OS-9® to run on the SH7750 board.

Figure 1-1. SH7750 Development Environment



Requirements and Compatibility

The following sections represent the host and target requirements for using the SH7750SE01 board.

Host Hardware Requirements (PC Compatible)

Your host PC should have the following minimum hardware characteristics:

- 32MB of RAM
- An Ethernet network card
- A PCMCIA card reader/writer

Host Software Requirements (PC Compatible)

Your host PC must have the following software installed:

- Microware OS-9 for SH4
- Microsoft® Windows 95, 98, ME, 2000, or NT

Target Hardware Requirements

Your SuperH evaluation board requires the following hardware:

- A power supply
- An RS-232 null modem serial cable
- An Ethernet cable (for connecting to an Ethernet network)
- VGA display and serial mouse (for use with MAUI®)

Target Hardware Setup

Refer to the Hitachi documentation for information on hardware setup prior to installing and configuring OS-9 on your SuperH evaluation board.

Settings

The factory default settings for the DIP switches and jumpers may not work with OS-9. Be sure the DIP jumpers and switches agree with the following settings:

Jumpers

- J1 has pins 1 and 2 connected. (The default RTC power is from main power source--not CNB.)
- J2 has pins 1 and 2 open.
- J3 has pins 1 and 2 connected.

Switches

The switch settings described are not the factory defaults. They are the switch settings that must be used to ensure OS-9 will run properly on the reference board. If a switch setting is not specified, use the factory default setting.

SW3 has switches 2, 3, 6

set to ONSets clock mode, bus size and endian selection

SW3 has switch 1 set to OFF

SW4 has switches 3, 4 set to ON

sets board to boot from EPROM

SW5	used for setting IP address (most significant byte) ON = 0 and OFF = 1
SW6	used for setting IP address ON = 0 and OFF = 1
SW7	used for setting IP address ON = 0 and OFF = 1
SW8	used for setting IP address (most significant byte) ON = 0 and OFF = 1

This can also be used for setting the least significant byte of the board's Ethernet address, since the least significant byte is the same for both the IP address and the Ethernet address.

Within each general purpose switch, the bit order is as follows: bit 1 is switch 8 and bit 8 is switch 1.

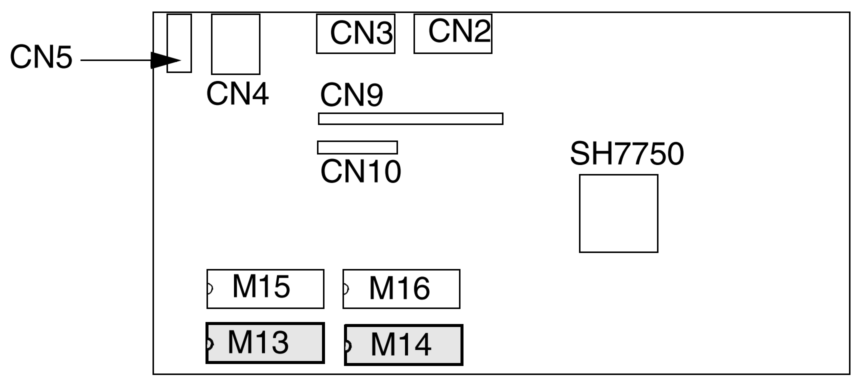
Installing the EPROM Devices

The first stage in configuring your SuperH evaluation board is to transfer the pre-configured coeboot image (included with your Microware OS-9 package) onto the EPROM devices on your evaluation board.

The coreboot image is generally responsible for initializing hardware devices and locating the high-level image as specified by its configuration, such as a FLASH part, hard disk, 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.

To install the EPROM devices, place them into sockets M13 (HIGH) and M14(LOW).

Figure 1-2. EPROM locations on the SuperH Evaluation Board



Refer to Placing a Coreboot Image in Flash Memory for information on programming a new coreboot image into the flash devices.

Connecting the Target to the Host

Connecting the SH7750SE01 to your host PC involves attaching the power, serial, Ethernet, and video cables to the solution engine. Once you have the board connected, you can use the serial console in Hawk™ to verify the serial connection.

Establishing a Serial Connection

To establish a serial connection, complete the following steps:

Step 1. Attach a 10BaseT Ethernet cable to connector CN4.

Step 2. Connect one end of a serial cable to connector CN2 on the SH7750SE01 board. Connect the other end to COM1 on the Host PC. Depending on your PC system, you may either need a straight or a reversed serial cable to make this connection. If you do not know what type of serial cable your machine uses, try a reversed cable first. If the connection fails (no boot messages appear in the communication program's window), try a straight serial cable.

Step 3. Insert the adapter plug into the power connector CN5 to apply power to the board.

Set Up HyperTerminal for Windows

To use Hyperterminal to connect your board and the host machine, complete the following steps:

Step 1. **For Windows 95/98:** From the Windows desktop, select `Start -> Programs -> Accessories -> HyperTerminal` to open HyperTerminal.

For Windows NT: From the Windows desktop, select `Start -> Programs -> Accessories -> HyperTerminal -> Hyperterminal` to start Hyperterminal.

Step 2. Enter a name for your HyperTerminal session.

Step 3. Select an icon for the new HyperTerminal session.

Step 4. **For Windows 95/98:** Click `OK`. The **Phone Number** dialog box appears.

For Windows NT: Click `OK`. The **Connect To** dialog box appears.

Step 5. **For Windows 95/98:** In the **Phone Number** dialog box, go to the **Connect Using** drop-down combo box and select the communications (COM) port that is connected to the SuperH evaluation board.

For Windows NT: In the **Connect To** dialog box, go to the **Connect Using** drop-down combo box and select the communications (COM) port that is connected to your SuperH evaluation board.

Step 6. Click `OK`. The **COM# Properties** dialog box appears (# represents the number of your chosen COM port, such as COM1).

Step 7. In the **Port Settings** tab, enter the settings as indicated in the following list:

- Bits per second: `9600`
- Data bits: `8`
- Parity: `None`
- Stop Bits: `1`
- Flow Control: `Xon/Xoff`

Step 8. Click OK. A connection should be established.

Step 9. Turn on the SuperH evaluation board. A boot menu similar to the one in the following illustration will appear after boot messages are displayed.

```
OS-9000 Bootstrap for the SuperH

ATA IDE disk found

Now trying to Override autobooters

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

Boot from PCMCIA PCCARD ---------------- <pcm_pc>
Boot embedded OS-9000 in-place --------- <bo>
Copy embedded OS-9000 to RAM and boot -- <lr>
Enter system debugger ------------------ <break>
```

Once you have connected the host system to the evaluation board, you can build a bootfile and place it on a PCMCIA IDE card, using the steps in the following sections.

Building the OS-9 Bootfile Image

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

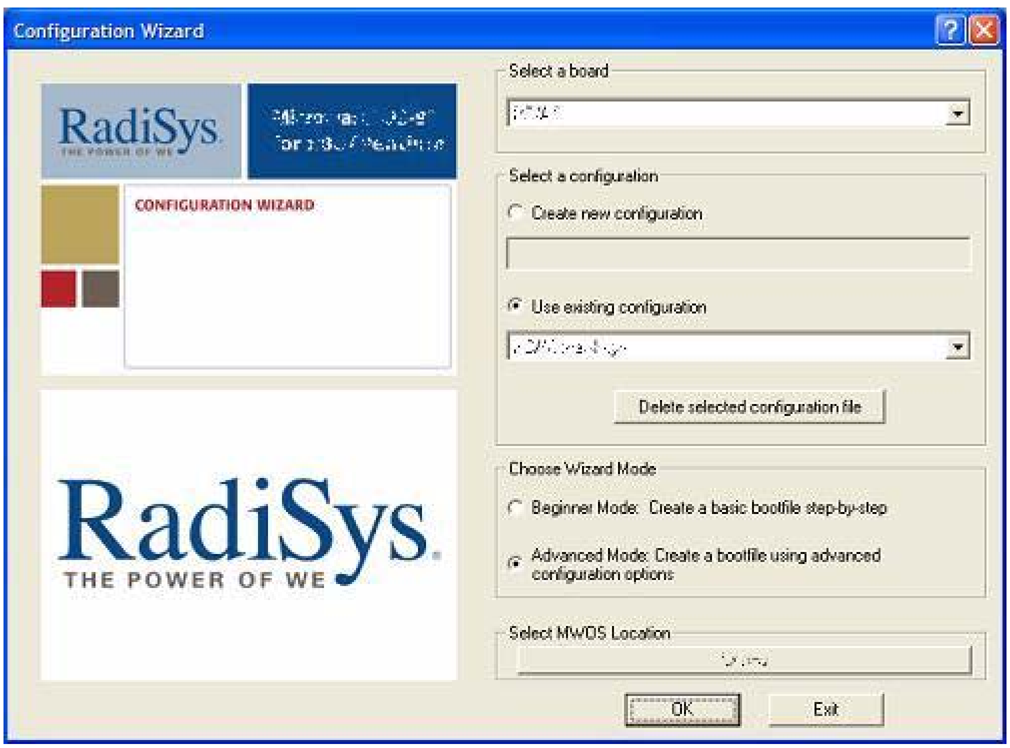
Starting the Configuration Wizard

The OS-9 coreboot image in the supplied coreboot image allows for booting from PCMCIA IDE cards. To boot from a PCMCIA IDE card, you need to place an OS-9 bootfile image on the card. The Configuration Wizard is used to create this bootfile image. The Configuration Wizard is a special purpose utility that simplifies the task of building OS-9 boot images. It is automatically placed on your host PC when you install one of the Microware OS-9 for SuperH packages.

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

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

Figure 1-3. Configuration Wizard Opening Screen

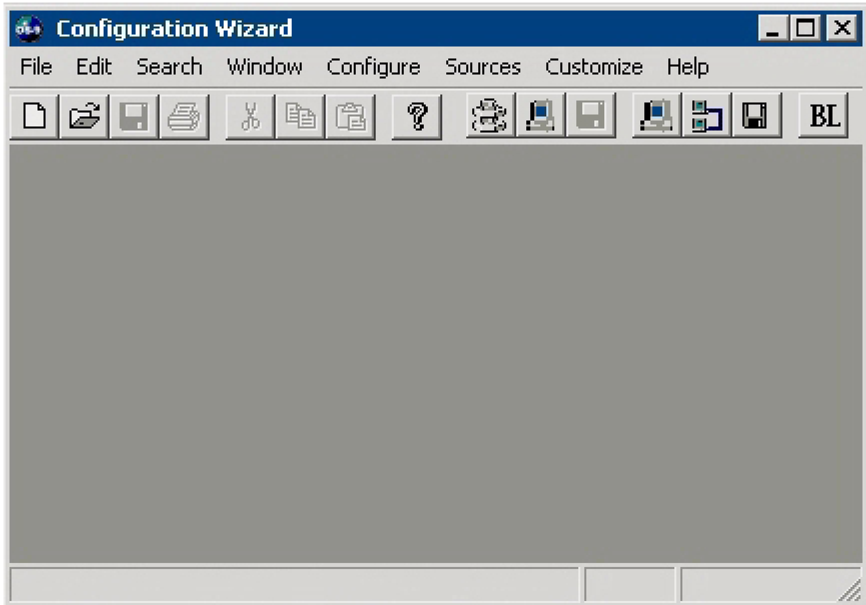


Step 2. Select your target board from the Select a board pull-down menu.

Step 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.

Step 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 1-4.

Figure 1-4. Configuration Wizard Main Window



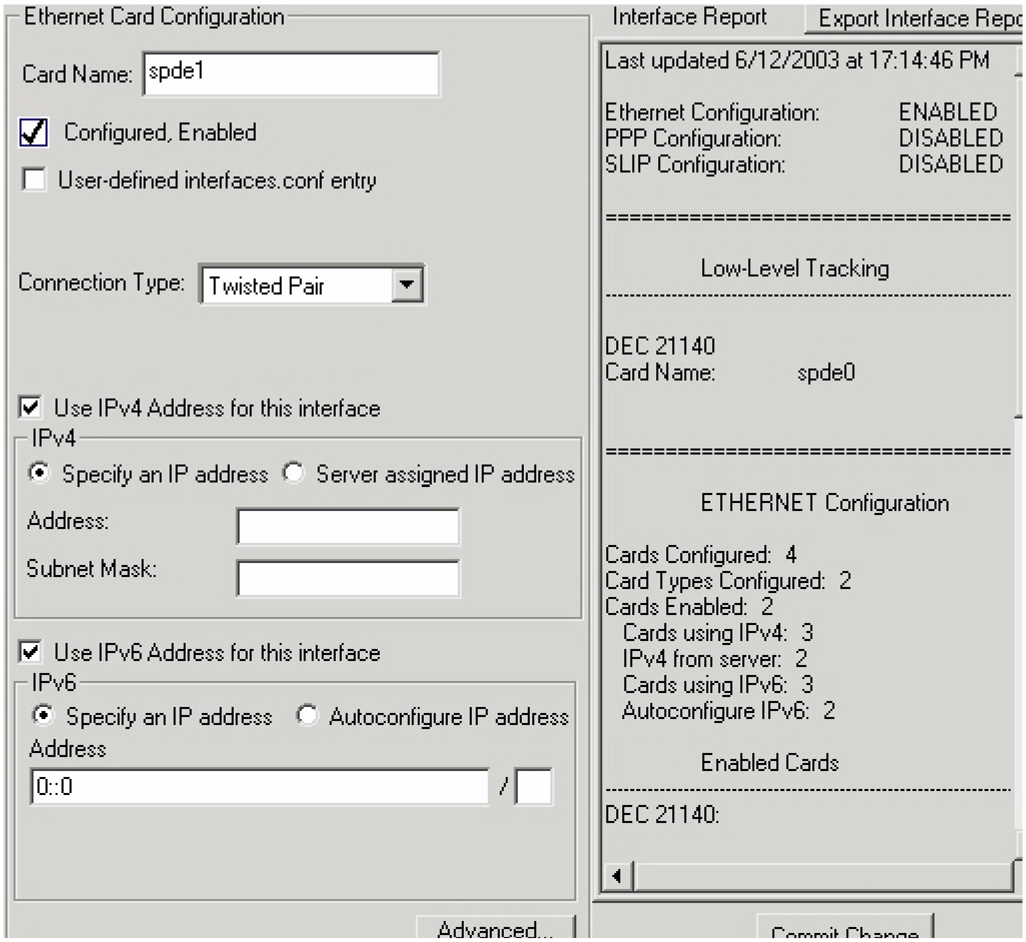
Building the Bootfile Image

Once in the Advanced mode of the Configuration Wizard, build a bootfile image by completing the following steps:

Step 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.

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 would like to enable IPv4 or IPv6 addressing. Figure 1-5 shows an example of this tab.

Figure 1-5. Interface Configuration



To learn more about IPv4 and IPv6 functionalities, refer to the *Using LAN Communications* manual.

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

Step 3. Select the `SoftStax® Setup` tab; select `Enable SoftStax`. Click `OK`.

Step 4. To enable the PCMCIA IDE function, select `System Disk Configuration` in the **Bootfile Configuration Buttons** group.

Step 5. Select `IDE Configuration`; select `Enable IDE Disk`. Click `OK`.

Step 6. Click the `Build Images` button to display the **Master Builder** window. (Do not change other settings at this point.)

Step 7. In the **Master Builder** window, select the following check boxes:

- **Disk Support**
- **Disk Utilities**
- **SoftStax® (SPF) Support**

- **User State Debugging Modules**

Step 8. Click Bootfile Only Image and click Build. This builds the bootfile image that can be placed on the PCMCIA IDE card.

Step 9. Insert the PCMCIA IDE card into the PCMCIA slot of your host computer. Click Save As to save the bootfile to the root directory of the PCMCIA IDE card. Name the bootfile os9kboot.

Step 10. Click Finish and select File -> Save Settings to save the configuration. Select File -> Exit to quit the Wizard.

Step 11. Make sure the power to the board is turned off and remove the PCMCIA IDE card from the host machine.

> If you insert a PCMCIA card into the PCMCIA socket of the SuperH evaluation board with power applied to the board, you will damage the PCMCIA card.

Step 12. Position the PCMCIA card so that the end with the PCMCIA female connector is facing the PCMCIA socket and the label on the top of the card is facing down.

Step 13. Slide the card into the socket until the card snaps onto the connector pins and the eject button pops out.

Step 14. Apply power to the board. The SH7709SE01 solution engine will boot to the mshell prompt, "$".

Step 15. To use Hawk to load and debug your applications, start the debugging daemons by typing the following commands:

```
spfndpd<>>>/nil&
ndpio<>>>/nil&
```

To perform system-state debugging, you will need to create a new coreboot image.

Creating a Startup File

When the Configuration Wizard is set to use a hard drive, or another fixed drive such as a PC Flash Card, as the default device, it automatically sets up the init module to call the startup file in the SYS directory in the target (For example: /h0/SYS/startup, /mhc1/SYS/startup). However, this directory and file will not exist until you create it. To create the startup file, complete the steps below:

Step 1. If not already available, create a SYS directory on the target machine where the startup file will reside (for example: makdir /h0/SYS).

Step 2. On the host machine, navigate to MWOS/OS9000/SRC/SYS. You should see the following files:

- motd: Message of the day file
- password: User/password file
- termcap: Terminal description file
- startup: Startup file

Below is the example startup file as it appears in this directory:

```
-tnxnp
tmode -w=1 nopause
*
*setime </term              ;* start system clock
setime -s                   ;* start system clock
link mshell csl;          * make "mshell" and "csl" stay in memory
* iniz r0 h0 d0 t1 p1 term  ;* initialize devices
* load utils                ;* make some utilities stay in memory
* tsmon /term /t1 &         ;* start other terminals
list sys/motd
setenv TERM vt100
tmode -w=1 pause
mshell<>>>/term -l&
```

Because the command lines in the startup file are system-dependent, it may be necessary to modify the file to fit your system configuration. It is recommended that you modify the file before transferring it to the target machine.

> Refer to the Making a Startup File section in Chapter 9 of the *Using OS-9* manual for more information on startup files.

Step 3. Transfer all files to the newly created `SYS` directory on the target machine. (You can use Kermit, or FTP in ASCII mode to transfer these files.)

Step 4. Since the files are still in DOS format, you will be required to convert them into the OS-9 format with the `cudo` utility. The following command is an example:

```
cudo -cdo password
```

This will convert the `password` file from DOS to OS-9 format.

> For a complete description of all the `cudo` command options, refer to the *Utilities Reference Manual* located on the Microware OS-9 CD.

2 Optional Procedures

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

These procedures involve customizing the coreboot image.The main reason for changing the coreboot image is to take advantage of ROM Ethernet services, such as System State Debugging. The System State Debugging limitation occurs because the IP address used in the EPROM image is set to `0.0.0.0`. If you want System State Debugging, you must create a new version of the coreboot image with an IP address assigned to the board.

If you are only doing User State Debugging under SoftStax, changing the coreboot image is not necessary.

The following sections are included:

- Placing a Coreboot Image in Flash Memory
- Placing a ROM Image into Flash Memory
- Programming the ROM Image into FLASH Memory
- Creating a Coreboot Image with an EPROM Programmer
- Making a ROM Image with an EPROM Programmer
- Compressing the Bootfile Image

Placing a Coreboot Image in Flash Memory

To place a coreboot image onto the SuperH board, you need to build the coreboot image, embed it in a bootfile, and program it into flash memory.

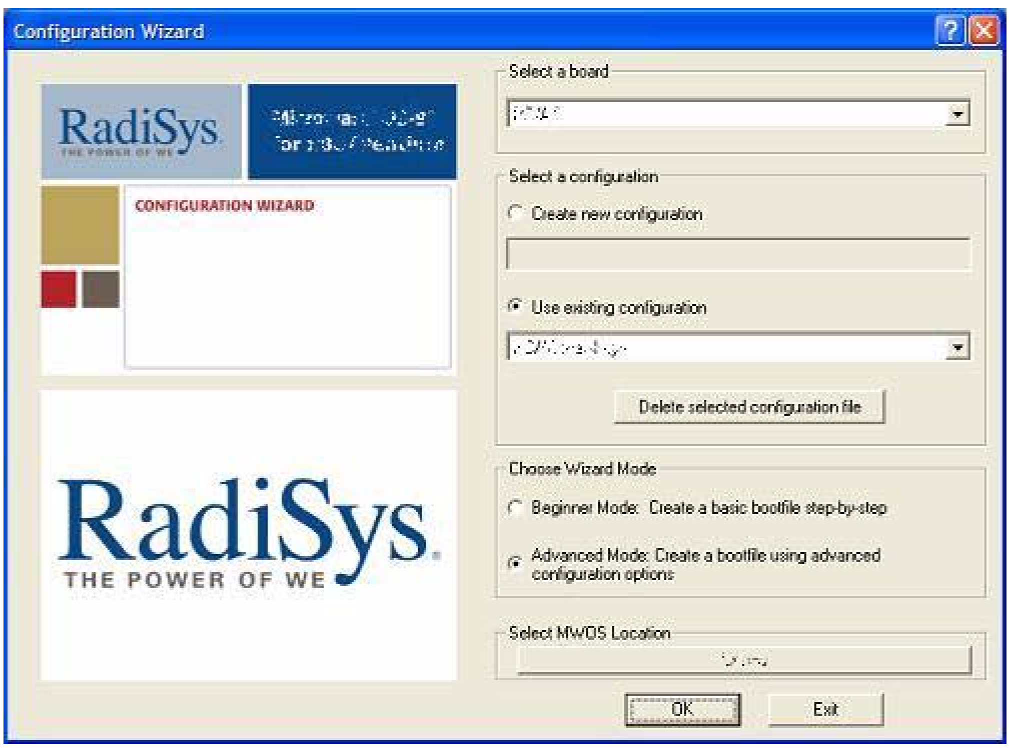
It is possible to create a new bootfile that can overwrite your current bootfile. To do this, you need to take steps to protect your current bootfile. To save the current bootfile, either change the current bootfile's name or move it to a subdirectory on the PCMCIA card.

Building the Coreboot Image

Complete the following steps to build the coreboot image:

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

Figure 2-1. Configuration Wizard Opening Screen

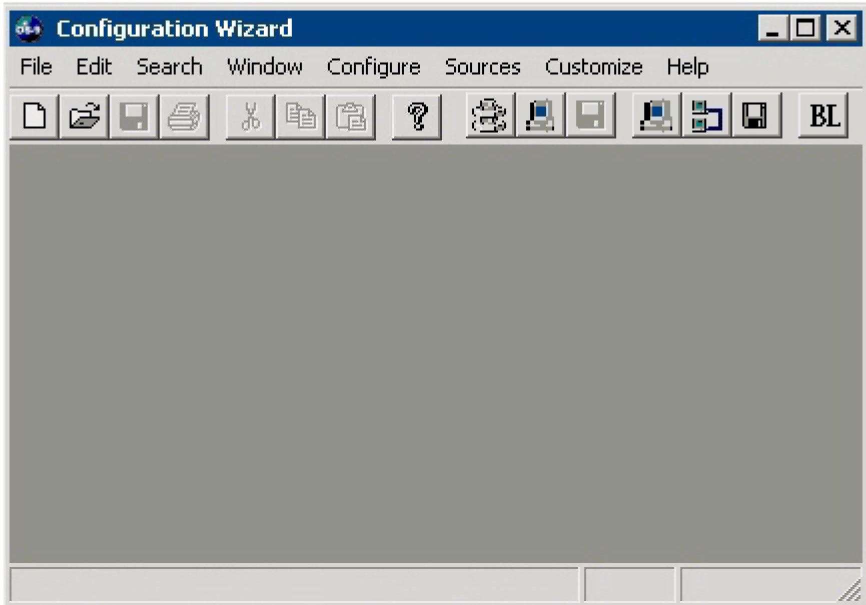


Step 2. Select your target board from the **Select a board** pull-down menu.

Step 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.

Step 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 2-2.

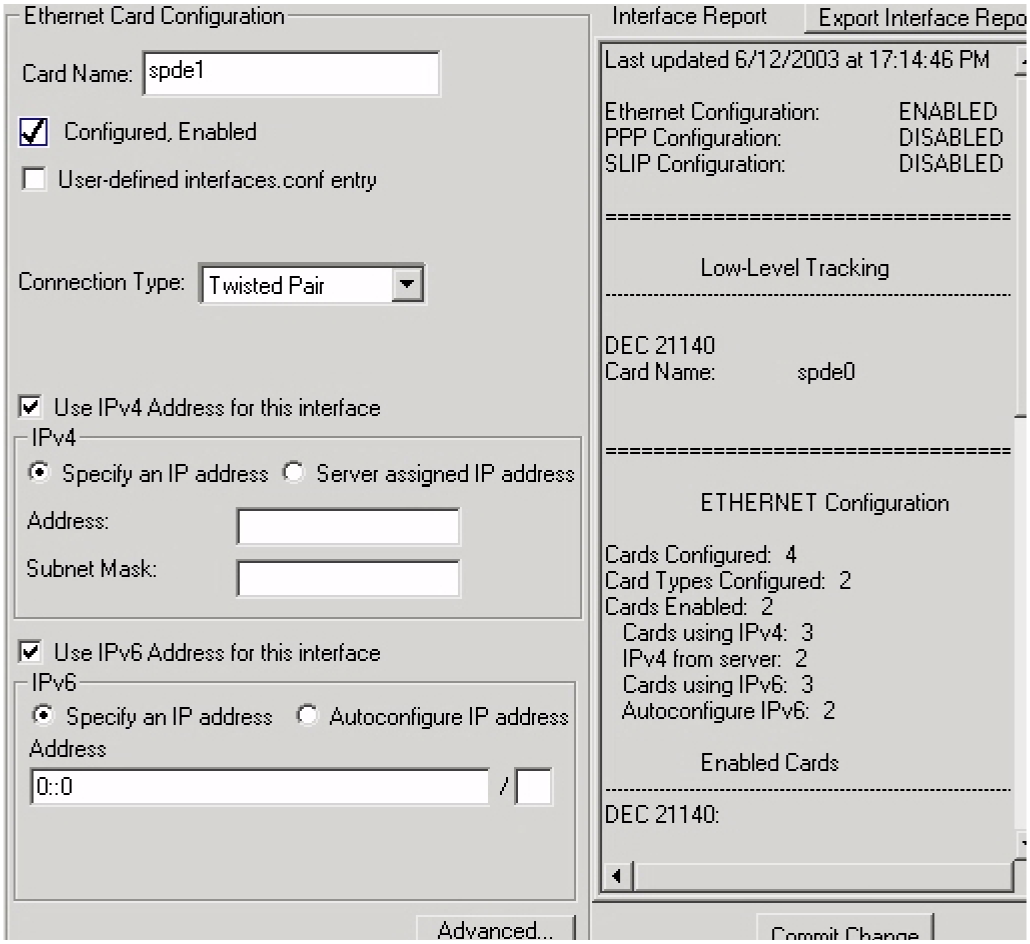
Figure 2-2. Configuration Wizard Main Window



Step 5. 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.

Step 6. 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 2-3 shows an example of the Interface Configuration tab.

Figure 2-3. Interface Configuration



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

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

Step 7. Click `OK` to close the window.

Step 8. Click `Build Images` to display the Master Builder window.

Step 9. Click `Coreboot Only Image` setting and click `Build`. The coreboot image is built.

Step 10. Click `Finish` to dismiss the Master Builder window.

Embedding the Coreboot Image in a Bootfile

To embed the coreboot image into your boot, complete the steps below:

Step 1. In the SuperH Configuration Wizard window, click the `Configure System Options` button. The Wizard's main window appears.

Step 2. Click on the `Bootfile Options` tab and select `PF-CORE`. PF-CORE includes the new coreboot image in the new bootfile as a data module. Click `OK`.

Step 3. Click `Build Images` to open the Master Builder window.

Step 4. Click `Bootfile Only Image` and click `Build`. The bootfile image is built, and the **Save As** button is enabled when the build is completed.

Step 5. Save the bootfile to the root directory of the PCMCIA IDE card. Use the name `os9kboot`.

Step 6. Click `Finish` to close the Master Builder window, and select `File` -> `Save Settings` to save the configuration.

Step 7. Select `File` -> `Exit` to quit from the Wizard.

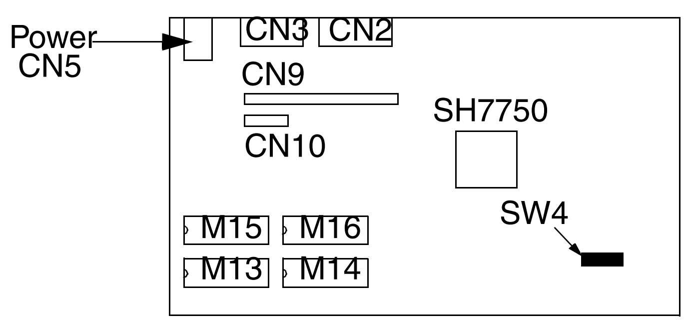
Writing the Coreboot Image into Flash Memory

To write the coreboot image into FLASH memory, complete the following steps:

Step 1. Remove power from the SuperH evaluation board.

Step 2. Locate the eight-switch dip switch labeled SW4 on the SuperH evaluation board.

Figure 2-4. Location of Switch 4 (SW4)



Step 3. Set switch SW4-3 (switch 3 on SW4) to the ON position. This tells the system to boot from the EPROM instead of the flash memory.

Step 4. Remove the PCMCIA IDE card containing `os9kboot` from the PC host and insert the card into the PCMCIA socket on the SuperH board.

Step 5. Open the Serial console in Hawk.

See Set Up HyperTerminal for Windows for more information on opening the Serial console.

Step 6. Apply power to the SuperH evaluation board. A boot menu similar to the following illustration appears:

```
OS-9000 Bootstrap for the SuperH
```

```
ATA IDE disk found

Now trying to Override autobooters

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

Boot from PCMCIA PCCARD ---------------- <pcm_pc>
Boot embedded OS-9000 in-place --------- <bo>
Copy embedded OS-9000 to RAM and boot -- <lr>
Enter system debugger ------------------ <break>
Restart the System --------------------- <q>
```

Step 7. Type pcm_pc to finish booting with the bootfile on the PCMCIA IDE card. The new bootfile containing the coreboot image is now loaded into the SuperH evaluation board's RAM memory. You are ready to load the coreboot image into flash ROM.

Step 8. At the shell prompt ($), type the following command: pflash.
This command erases flash memory, writes the new coreboot image into the flash memory and verifies the contents of the flash memory.

Step 9. When the shell prompt appears again, remove power from the SuperH evaluation board.

Step 10. Set switch 4-3 to the OFF position.

Step 11. Reboot the system. The SuperH board is now using the new coreboot image in flash memory.

Once you have completed these steps, restore your original bootfile by deleting the bootfile that was created in this section and replacing it with your original bootfile.

To make your original bootfile active, restart the system.

Placing a ROM Image into Flash Memory

To put a ROM image onto the SuperH board, you have to build the image, embed it in another bootfile to transfer it to the board, and store it in flash memory.

Once you place a ROM image into flash memory, you have the ability to boot to the shell from flash memory instead of from your PCMCIA IDE card. If you want this combined image to have the same bootfile settings you currently use, start the Wizard with the configuration name under which you saved those settings.

Step 1. Open the Configuration Wizard as described in the Starting the Configuration Wizard section.

Step 2. 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.

Step 3. Select any other coreboot or bootfile options you want included in your ROM image, then open the Master Builder window from the Wizard menu and select Coreboot + Bootfile and click Build. The ROM image is built and saved.

Step 4. Click the Finish button to close the Master Builder window.

Embedding the ROM Image in a Bootfile

To embed the ROM image into your boot, complete the following steps:

Step 1. In the SuperH Configuration Wizard window, click `Configure System Options`. The SH4:<your configuration name> window appears.

Step 2. Click on the `Bootfile Options` tab.

Step 3. Click `PF-ROM`. PF-ROM will include the ROM image in the new bootfile as a data module.

Step 4. If there are any other bootfile options you want active at this time, select them as well.

Step 5. Click `OK` to close the window.

Step 6. Click `Build Images` to open the Master Builder window.

Step 7. Click `Bootfile Only Image` and then click `Build`. The bootfile containing the ROM image is built and saved. The Save As button is enabled when the build is completed.

Step 8. Save the bootfile to the root directory of the PCMCIA IDE card. Use the name `os9kboot`.

Step 9. Click `Finish` to close the Master Builder screen, and select `File -> Save Settings` to save the configuration.

Step 10. Select `File -> Exit` to quit from the Wizard.

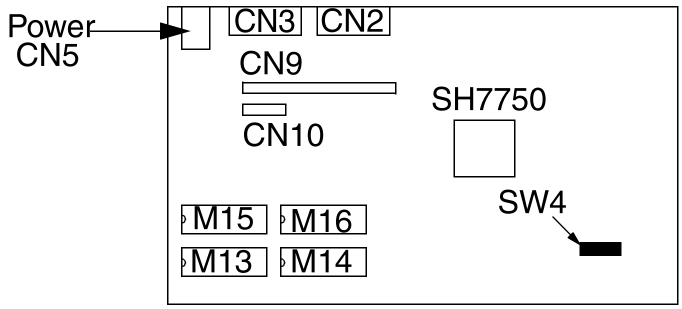
Programming the ROM Image into FLASH Memory

To program the ROM image into Flash memory, complete the following steps:

Step 1. Remove power from the SuperH evaluation board.

Step 2. Locate the eight-switch dip-switch labeled SW4 on the SuperH evaluation board.

Figure 2-5. Location of Switch 4



Step 3. Set switch SW4-3 (switch 3 on SW4) to the ON position. This tells the system to boot from the EPROM instead of the flash memory.

Step 4. Remove the PCMCIA IDE card from the PC host and insert the card into the PCMCIA socket on the SuperH board.

Step 5. Open the Serial console in Hawk. Refer to Set Up HyperTerminal for Windows for more information on opening the Serial console.

Step 6. Apply power to the board. A boot menu appears, similar to that shown below:

```
OS-9000 Bootstrap for the SuperH

ATA IDE disk found

Now trying to Override autobooters

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

Boot from PCMCIA PCCARD ---------------- <pcm_pc>
Boot embedded OS-9000 in-place --------- <bo>
Copy embedded OS-9000 to RAM and boot -- <lr>
Enter system debugger ------------------ <break>
Restart the System --------------------- <q>

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

Step 7. Type `pcm_pc` to finish booting. The new bootfile containing the ROM image is loaded into the SuperH evaluation board's RAM memory; you are ready to load the ROM image into flash ROM.

Step 8. At the shell prompt (`$`), type the following command:

```
pflash
```

The `pflash` command erases flash memory, writes the new ROM into the flash memory, and verifies the contents of the flash memory.

Step 9. When you get the shell prompt again, turn off the SuperH evaluation board.

Step 10. Set switch SW4-3 to the OFF position.

Step 11. Restart the system and enter `<bo>` at the Boot Menu. The SuperH board will boot to the shell using the new ROM image in flash memory.

Creating a Coreboot Image with an EPROM Programmer

This section instructs you on how to create the coreboot image through the point where you transfer the file to your EPROM programmer. Refer to the instructions for your EPROM programmer to learn how to program the new coreboot image into the EPROMS.

Step 1. Open the Configuration Wizard as described in the Starting the Configuration Wizard section.

Step 2. 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.

Step 3. Select any other coreboot options you want included in your coreboot image.

Step 4. Select `Configure -> Build Image` to display the Master Builder window.

Step 5. Click `Coreboot Only Image` and click `Build`.

Step 6. Click `Save As` to save the coreboot image to a directory of your choosing. The default file name is `coreboot`.

Step 7. Click `Finish` to close the Master Builder window, and select `File -> Save Settings` to save the configuration.

Step 8. Select `File -> Exit` to quit from the Wizard.

Step 9. Transfer the coreboot image to the EPROMs with the EPROM programmer. You will need to follow the documentation for the EPROM programmer to complete this step.

Step 10. With the power to the board turned off, insert the EPROMs into the SuperH board.

Step 11. Set SW4-3 (switch 3 on SW4) to the ON position so the board will boot from the EPROMs.

Step 12. Turn on power to the board. A boot menu appears, similar to that shown below:

```
OS-9000 Bootstrap for the SuperH

ATA IDE disk found

Now trying to Override autobooters

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

Boot from PCMCIA PCCARD ---------------- <pcm_pc>
Boot embedded OS-9000 in-place --------- <bo>
Copy embedded OS-9000 to RAM and boot -- <lr>
Enter system debugger ------------------ <break>
```

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

Step 13. Select the booting method you want to use to boot the system to the shell prompt.

Making a ROM Image with an EPROM Programmer

The following steps detail how to create the ROM image through the point where you transfer the file to your EPROM programmer. Refer to your EPROM programmer's instructions to learn how to program the new ROM image into the EPROMS.

Step 1. Open the Configuration Wizard as described in the Starting the Configuration Wizard section.

Step 2. 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.

Step 3. Select the coreboot and bootfile options you want included in your ROM image.

Step 4. Select `Configure -> Build Image` to display the Master Builder window.

Step 5. Click `Coreboot+Bootfile Image`, and click `Build`.

Step 6. Make sure the ROM image is not larger than your available EPROM memory. If it is too big, you complete one of the following three actions:

- select the Pack ROM option
- turn off some of the bootfile options
- select the Pack ROM option and deselect some bootfile options

Step 7. Click `Save As` to save the ROM image to a directory of your choosing. If you do not have that directory on the drive, you can create it.

Step 8. Click `Finish` to close the Master Builder window.

Step 9. Select `File -> Save Settings`, and `File -> Exit` to close Configuration Wizard.

Step 10. Transfer the ROM image to the EPROMS with the EPROM programmer. You will need to follow the documentation for the EPROM programmer to complete this step.

Step 11. With the power to the board turned off, insert the EPROMS into the SuperH board.

Step 12. Set switch 4-3 (switch 3 on SW4) to the ON position so the board will boot from the EPROMS.

Step 13. Turn on power to the board. A boot menu appears, similar to the example below:

```
OS-9000 Bootstrap for the SuperH

ATA IDE disk found

Now trying to Override autobooters

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

Boot from PCMCIA PCCARD ---------------- <pcm_pc>
Boot embedded OS-9000 in-place --------- <bo>
Copy embedded OS-9000 to RAM and boot -- <lr>
```

```
Enter system debugger ------------------ <break>
Restart the System --------------------- <q>
```

Step 14. Select the booting method you want to use to boot the system to the shell prompt.

Compressing the Bootfile Image

OS-9 bootfiles can be compressed to allow more modules to be loaded into a bootfile; this can be useful if you plan on storing your image on a small FLASH part or a floppy disk.

The bootfile compression utility performs the compression at approximately a 2.5:1 ratio.

Complete the following steps to compress your image:

Step 1. Verify that your coreboot contains the `uncompress` module. This module can be found in the pre-built ROM and coreboot images that were shipped with your Microware OS-9 product.

The `uncompress` module must be included in order for the compression to execute properly.

Step 2. Open the Configuration Wizard and select `Configure -> Coreboot -> Main Configuration` from the main menu.

Step 3. Select the `Bootfile Compression` tab. Verify that the **Include bootfile uncompress module** box is checked and select `OK`.

Step 4. When you are ready to build the image, open the **Master Builder** dialog. Verify that the **Compress Bootfile** box is checked and then press `Build` to begin the installing the image.

3 Board-Specific Reference

This chapter represent board specific considerations for using Microware OS-9 for SH4. The following sections are included:

- The Fastboot Enhancement
- Enabling PCMCIA IDE Interrupts

The Fastboot Enhancement

The Fastboot enhancements to OS-9 were added to address the needs of embedded systems that require faster system bootstrap performance than normal. OS-9's normal bootstrap performance is mostly attributable to its flexibility. OS-9 can handle 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 allow the system to perform a minimal amount of runtime configuration. As a result, a significant increase in bootstrap speed is achieved.

Overview

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

One very important feature of the Fastboot enhancement is that not only can the control flags be statically defined when the embedded system is initially configured, but they may also be 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.

Also, the Fastboot enhancement's versatility allows for special considerations under certain circumstances. This versatility would be useful in a system where normally 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 could respond to some form of user input that would inform it that additional checking and system verification is desired.

Implementation Overview

The Fastboot configuration flags have been implemented as a set of bit fields. An entire 32-bit field has been dedicated for bootstrap configuration. This four-byte field is contained within the set of data structures shared by the ModRom sub-components and the kernel. Hence, the field is available for modification and inspection by the entire set of system modules (high-level and low-level). Currently, there are just six bit flags defined with eight bits reserved for user-definable bootstrap functionality. The reserved user-definable bits are the high-order eight bits (31-24). This leaves bits available for future enhancements. The currently defined bits and their associated bootstrap functionality are listed below:

B_QUICKVAL

The B_QUICKVAL bit indicates that only the module headers of modules in ROM are to be validated during the memory module search phase. This causes the CRC check on modules to be omitted. This option is potentially a large time saver due to the complexity and expense of CRC generation. If a system has many modules in ROM, where access time is typically longer than RAM, omitting the CRC check on the modules will drastically decrease the bootstrap time. It is fairly rare that corruption of data occurs in ROM. Therefore, omitting CRC checking will usually be a safe option.

B_OKRAM

The B_OKRAM bit informs both the low-level and high-level systems that they should accept their respective RAM definitions without verification. Normally, the system probes memory during bootstrap based on the defined RAM parameters. This allows system designers to specify a possible RAM range which the system will validate upon startup. Thus the system can accommodate varying amounts of RAM. But 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 just like the B_OKRAM option except that it applies to the acceptance of the ROM definition.

B_1STINIT

The B_1STINIT bit causes acceptance of the first `init` module found during cold-start. By default, the kernel searches the entire ROM list passed up by the ModRom for `init` modules before it accepts and uses the `init` module with the highest revision number. In a statically defined system, a good deal of time can be saved by using this option to omit the extended `init` module search.

B_NOIRQMASK

The B_NOIRQMASK bit informs the entire bootstrap system that it should not mask interrupts for the duration of the bootstrap process. Normally, the ModRom code and the kernel cold-start mask interrupts for the duration of the system startup. But some systems that have a well defined interrupt system (i.e. completely calmed by the `sysinit` hardware initialization code) and also have a requirement to respond to an installed interrupt handler during system startup can enable this option to prevent the ModRom and the kernel cold-start from disabling interrupts. This is particularly useful in power-sensitive systems that need to respond to "power-failure" oriented interrupts. Some portions of the system may still mask interrupts for short periods during the execution of critical sections.

B_NOPARITY

If the RAM probing operation has not been omitted, the B_NOPARITY bit causes the system to not perform parity initialization of the RAM. Parity initialization occurs during the RAM probe phase. The B_NOPARITY option is useful for systems that either require no parity initialization at all or systems that only require it for "power-on" reset conditions. Systems that only require parity initialization for initial "power-on" reset conditions can dynamically use this option to prevent parity initialization for subsequent "non-power-on" reset conditions.

Implementation Details

This section describes the compile-time and runtime methods by which users can control the bootstrap speed of their 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. Users can redefine the macro for recompilation to create a new bootstrap configuration. The new over-riding value of the macro should be established by redefining the macro in the `rom_config.h` header file or as a macro definition parameter in the compilation command.

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

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

And here 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 would result in a bootstrap method which would accept the RAM and ROM definitions as they are without verification, and also validate modules solely on the correctness of their module headers.

Runtime Configuration

The default bootstrap configuration can be overridden at runtime by changing the `rinf->os->boot_config` variable from either a low-level P2 module or from the `sysinit2()` function of the `sysinit.c` file. The runtime code can query jumper or other hardware settings to determine what user-defined bootstrap procedure should be used. An example P2 module is shown below.

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;
}
```

Enabling PCMCIA IDE Interrupts

Due to a problem with losing interrupts when using certain PCMCIA IDE cards with the SuperH (SH7750) board, the default configuration of OS-9 has been set to polled mode for accessing PCMCIA IDE type devices.

The following PCMCIA IDE cards are known to have problems with interrupts:

- the SanDisk PCMCIA PC CARD ATA 4MB card
- the SanDisk PCMCIA PC CARD ATA 20MB card

The following PCMCIA IDE cards have not shown any problems with interrupts:

- the Viking PCMCIA PC CARD ATA 12MB card
- the EXP Disk Traveler HDG 1.4GB card
- the Maxtor Hard Card series

All of the above cards (including the SanDisk cards) will work with polled mode. If you need to enable interrupts for use with your applications, you will need to follow the steps outlined in Enabling PCMCIA IDE Interrupts on the SuperH.

If you are only creating a bootfile image, select `llcis` from the Bootfile Options tab in the wizard to load the low-level Microware Socket Services. When using this option, you do not need to re-create the flash `coreboot` image.

Before You Begin

You need to test to see if your PCMCIA IDE card will work with PCMCIA interrupts enabled. If the following sequence of three commands work, then you can safely enable interrupts on your system.

```
$chd /mhc1
$save kernel
$ident kernel
```

Enabling PCMCIA IDE Interrupts on the SuperH

To enable interrupts on PCMCIA IDE devices the Microware Socket Services and device descriptors must be updated.

The Microware PCMCIA Socket Services are included in a p2module called `llcis` as well as in the `pcmcia` utility's module. Both of these modules should be compiled with interrupts enabled to use PCMCIA IDE interrupts.

Updating the llcis Module

Update the makefile for the `llcis` module by completing the following steps:

Step 1. Change to the `LLCIS` directory. The `LLCIS` directory is found in the following path:
`MWOS\OS9000\SH4\PORTS\SH7750SE\ROM\LLCIS`.
`LLCIS` contains a file named `makefile`.

Step 2. Using a text editor, open `makefile`.

Step 3. Remove the '#' character from the following line:
`SPEC_COPTS = -dSINGLE_SOCKET # -dUSE_IRQ`

Step 4. Type `os9make` from the `LLCIS` directory to build a new `llcis` module.

Updating the PCMCIA Utility

After you update the makefile for the `llcis` module, you need to update the makefile for the PCMCIA utility. The path to the PCMCIA utility's makefile is as follows:

`MWOS\OS9000\SH4\PORTS\SH7750SE\UTILS\PCMCIA\makefile.`

Step 1. Change to the PCMCIA directory.

Step 2. Using a text editor, open the file named `makefile`.

Step 3. Remove the '#' character from the following line:

Step 4. `SPEC_COPTS =    -dSINGLE_SOCKET -k # -dUSE_IRQ`

Step 5. Type `os9make` from the PCMCIA directory to build a new `pcmcia` module.

Updating the RBF/PCF PCMCIA IDE Device Descriptors

After you update the modules `llcis` and `pcmcia`, you need to update the PCMCIA IDE device descriptors. The PCMCIA IDE device descriptors are found in the `config.des` file. The path to the `config.des` file is as follows:

`MWOS\OS9000\SH4\PORTS\SH7750SE\RBF\RB1003\config.des`

Step 1. Change to the `RB1003` directory.

Step 2. Using a text editor, open the file named `config.des`.

Step 3. Find the following section of code in the file:

```
init dev_specific {
   ds_idetype = IDE_TYPE_PCMCIA;
   ds_polled = IDE_POLLED;
   ds_altstat = HD_ALTSTAT;
   ds_timeout = 30;
};
```

Step 4. Change `IDE_POLLED` to `IDE_INTERRUPTS` in the following line:

```
ds_polled = IDE_POLLED;
```

Step 5. Save your changes to the `config.des` file and change to the following directory: `MWOS\OS9000\SH4\PORTS\SH7750SE\RBF\RB1003\DESC`

Step 6. Type `os9make`. This will build the RBF descriptors.

Step 7. You have now enabled OS-9 to use PCMCIA IDE interrupts with the SH7750SE. You should now create a new build using the Configuration Wizard.

A Board-Specific Modules

This appendix provides a list of the SuperH hardware support devices and gives an alphabetical listing of the coreboot and bootfile modules. The following sections are included:

- SuperH Hardware Support Devices
- Low-Level System Modules List
- High-Level System Modules List

SuperH Hardware Support Devices

The following sections provide a list of the SuperH hardware support devices, including modules and descriptors; the modules and descriptors for each support device are found in the following location:

`MWOS\OS9000\SH4\PORTS\SH7750SE\CMDS\BOOTOBJS`

PIC Support

Module

Not available

PCMCIA Support for IDE Type Devices

Module

`rb1003`

Descriptors

`/hc1.h0`	PCMCIA RBF type device, primary master partition #1 *
`/hc1fmt`	PCMCIA RBF type device, primary master partition 1, format enabled *
`/hcfmt`	PCMCIA RBF type device, primary master entire disk, format enabled *
`/mhc1`	PCMCIA PC type device, primary master partition 1
`/mhc1.h0`	PCMCIA PC type device, primary master partition 1, default device
`/mhc1fmt`	PCMCIA PC type device, primary master, partition #1, format enabled
`/mhcfmt`	PCMCIA PC file system type device, primary master entire disk, format enabled

The Configuration Wizard does not support using a PCMCIA IDE card with RBF. If an item above is marked with an "*", it is assumed that the PC file system will be used.

Super I/O Support for IDE Type Devices

Module

`rb1003sio`

Descriptors

`/hcsio1.h0`	Super I/O RBF type device, partition #1 *
`/hcsio1fmt`	Super I/O RBF type device, primary master partition 1, format enabled *
`/hcsiofmt`	Super I/O RBF type device, primary master entire disk, format enabled *
`/mhcsio1`	Super I/O PC file system type device, primary master partition 1

/mhcsio1.h0 Super I/O PC file system type device, primary master partition 1, default device

/mhcsio1fmt Super I/O PC file system type device, primary master partition #1, format enabled

/mhcsiofmt Super I/O PC file system type device, primary master entire disk, format enabled

Super I/O Support for PS/2 Type Devices

Module

sc8042k

Descriptor

/kxo keyboard descriptor

/m0 mouse descriptor

Super I/O Support for Parallel Port

Module

scp87303

Descriptor

/p printer descriptor

Real-Time Clock

Module

rtc7750

Power Management Extension

Module

pwrext

Ticker (System Clock) Support

Module

tk7750

Serial Support

Module

sc7750

Descriptors /term /t1

`t1` is assigned to the SCIF port of the 7750 internal UART. The connector is located at the rear of the board near the Ethernet connector. It is labeled as `CN2 SH7750 SCIF`.

`t1`: serial port #1

Driver Name: `sc7750`

Default Baud Rate: 9600

Default Parity: None

Default Data Bits: 8

Software/Hardware/Auto handshaking is supported.

To use it: Select `scif7750 p1` in the Configuration Wizard.

Module

`scscish4`

Descriptors /t2

`t2` is assigned to the SCI port of the 7750 internal UART. The connector is located on the HY7709PCHK-I/O expansion board. It is labeled as RS232 Ch1 CN3.

`t2`: serial port #2

Driver Name: `scscish4`

Default Baud Rate 9600

Default Parity: None

Default Data Bits: 8

To use it: Select `sci7750 P1` in the Configuration Wizard.

Baud Rates

The following OS-9 baud rates are supported by the `sc7750` and `scscish4` drivers:

Table A-1. Supported SC7750, scscish4 Baud Rates

50	75	110	134.5	150	300
600	1200	1800	2000	2400	3600
4800	7200	9600	19200	38400	

The following OS-9 baud rates are not supported by the sc7750 driver:

Table A-2. sc7750, scscish4 Baud Rates Not Supported

31250	56000	57600	64000	115200

Module

`sc16550`

Descriptors /term /t3

`t3` is assigned to the SMC 37C935 16550 (compatible UART). The connector is located on the side of the board near the Ethernet connector. It is labeled as `CN3 COM1`.

`t3`: serial port #3

Default Baud Rate: 9600

Default Parity: None

Default Data Bits: 8

To use it: Select `16550 p1` in the Configuration Wizard.

Module

`sc16550`

Descriptors /term /t4

`t4` is assigned to the SMC 37C935 16550 (compatible UART). The connector for t4 is located on an expansion board.

`t4`: serial port #4

Default Baud Rate: 9600

Default Parity: None

Default Data Bits: 8

To use it: Select `16550 p2` in the Configuration Wizard.

Baud Rates

The following OS-9 baud rates are supported by the `sc16550` driver:

Table A-3. Supported sc16550 Baud Rates

50	75	110	134.5	150	300
600	1200	1800	2000	2400	3600
4800	7200	9600	19200	38400	

The following OS-9 baud rates are not supported by the sc16550 driver:

Table A-4. sc16550 Baud Rates Not Supported

31250	56000	57600	64000	115200

RAM Disk Support

Module

`ram`

Descriptors

`/r0` default size 512k *

`/r0.dd`

* The Configuration Wizard allows you to set the size of the RAM disk.

Optional Serial Support

Optional serial support using polled serial driver services via ROM driver is available.

Module

`scllio` low-level serial driver

Descriptors

`/term` device descriptor for using the low-level console for high-level I/O

Init Modules

`configurer` self-configured `init` module created by Configuration Wizard

`nodisk` standard `init` module with no initial device

MAUI Hardware Support Modules

Module

`gx_ygv618` Yamaha YGV618 graphic device driver

Descriptor

`gfx` Yamaha YGV618 graphics device descriptor

Module

`gx_hd66420` LCD HD66420 graphics device driver

This module is not currently supported.

Descriptor

`gfx_lcd` LCD HD66420 graphics device descriptor

This descriptor is not currently supported.

SPF Hardware Support Modules

Module

`sphdlc` HDLC framer driver

Descriptor

`hdlc0` HDLC device descriptor

Module

`spslip` SLIP protocol driver

Descriptor

`sps10` SLIP dd

Module

`sp7750` Ethernet driver for the National DP83902 chip

Descriptor

`spne0` DP83902 Ethernet device descriptor

Low-Level System Modules List

Table A-5 lists all of the coreboot modules available for the SH7750SE Reference platform. The list is organized alphabetically. The modules are not placed in the coreboot file in this order, and each module is not necessarily used in every build.

Table A-5. Coreboot Image Modules

Module	Description
`bootsys`	This is a module that provides booter services.
`cnfgdata`	This is the standard `cnfgdata` module (data module containing configuration parameters).
`cnfgdata_ configurer`	This is a `cnfgdata` module (data module containing configuration parameters) created by Configuration Wizard.
`cnfgfunc`	This is a module that retrieves configuration parameters from the `cnfgdata` module.
`commcnfg`	This is a module that retrieves the name of the low-level auxiliary communication port driver from the `cnfgdata` module.
`conscnfg`	This is a module that retrieves the name of the low-level console driver from the `cnfgdata` module.
`console`	This is a provides high-level I/O hooks into low-level console serial driver.
`dbgentry`	This is a module that provides hooks to low-level debugger server.
`dbgserv`	This is a debugger server module.
`excption`	This is a low-level exception services module.
`fdman`	This is the RBF (Random Block File) floppy and IDE drive manager. (RBF is the native OS-9 file system.)

Table A-5. Coreboot Image Modules (Continued)

Module	Description
flshcach	This is a module that provides the low-level cache flushing routine
ide	This is a low-level IDE booter module.
initext	This is a user-customizable system initialization module.
ioscifsh7750	This is a low-level serial driver for SH7750 SCIF serial port.
ioscish7750	This is a low-level serial driver for SH7750 SCI serial ports.
io16550	This is a low-level driver for 16550-compatible UART serial ports.
ll83902	This is a low-level Ethernet driver module.
llbootp	This is a low-level BOOTP booter module.
llcis	This is a low-level PCMCIA configuration information service module.
llip	This is a low-level IP protocol module.
llkermit	This is a low-level Kermit protocol module.
llslip	This is a low-level SLIP protocol module.
lltcp	This is a low-level TCP protocol module.
lludp	This is a low-level UDP protocol module.
notify	This is a module that coordinates use of low-level I/O drivers in system and user-state debugging.
override	This is a target-independent booter module that enables overriding of the autobooter. If the space bar is pressed within three seconds after booting the target, a boot menu is displayed. Otherwise, booting proceeds with the first autobooter.
parser	This is a `parser` is called by the booters to parse the key fields from the `cnfgdata` module and the user input (user parameter fields) during system boot.
pcman	This is a PCF (PC File) floppy and IDE drive manager.
portmenu	This retrieves a list of configured booter names from the ROM `cnfgdata` module.
protoman	This is a low-level protocol manager module.
restart	This is a booter module that restarts boot process.
romboot	This is a booter module that locates the OS-9 bootfile stored in ROM, FLASH, or NVRAM.
rombreak	This is a booter module that enables the break option in the boot menu (used to enter the debugger module).
RomBug	This is a ROM monitor debugger client module.
romcore	This is a bootstrap code for Hawk IDE.
sh4timer	This is a simulated low-level timer module.
sndp	This is a system state debug client module.
usedebug	This is a debugger configuration module.

High-Level System Modules List

Table A-6 lists all of the bootfile modules available for Microware OS-9 for SuperH. The list is organized alphabetically. The modules are not placed in the bootfile in this order, and each module is not necessarily used in every build.

Table A-6. Bootfile Image Modules

Module, Device Driver, File Manager, or Descriptor	Description
abort	This abort the switch handler.
activ	This is a module containing the activ system command. activ activates processes that were stopped by suspend.
alias	This is a utility that assigns an alternate name to a device pathlist.
aloha	This is a MAUI demonstration program that draws text and receives input.
arp	This is the arp utility displays and modifies the Internet-to-Ethernet address translation tables used by the address resolution protocol (ARP).
attr	This is a utility that examines or changes the security attributes (<permissions>) of the specified file(s).
backup	This is the backup utility copies all data from one device to another.
bfed	This is a screen-oriented binary file editor utility.
binex	This utility converts binary files to S-record files.
bootpd	This is a module that is the server daemon handling client BOOTP requests.
bootgen	This is a utility that builds and links a bootstrap file.
break	This is a module containing the break basic system command. break executes a system call that stops the operating system and all user processes and returns control to the ROM debugger.
build	This is a utility that builds a text file from standard input.
cache	This is a module that enables the data cache.
cdb	This is a default MAUI Configuration Description Block for Yamaha YGV618 daughter board
cdb_lcd	This is a MAUI Configuration Description Block for LCD HD66420 daughter board. This item is not currently supported.
cfp	This is the utility that creates a temporary procedure file in the current data directory and then invokes the shell to execute it.
chown	This is the utility that changes the owner ID of a file or directory to the owner ID specified.

Table A-6. Bootfile Image Modules (Continued)

Module, Device Driver, File Manager, or Descriptor	Description
cmp	This is the utility that opens two files and performs a comparison of the binary values of the corresponding data bytes of the files.
code	This is the utility that prints the input character followed by the hex value of the input character.
compress	This is the utility that reads the specified text file(s), converts it to a compressed form, and writes the compressed text file to standard output or to an optional output file.
configurer	This is the OS-9 initialization module.
copy	This is the utility that copies data from one file to another file.
csl	This is the C shared library module.
count	This is the utility that counts the number of lines in a file. Options include character count and word count.
date	This is the module containing the date basic system command. date displays the current system date and time.
dcheck	This is the utility that detects the integrity of the directory and file linkages of a disk device.
default.fnt	This is a module containing MAUI default fonts. Used by the hello and aloha demo programs.
deiniz	This is the utility that removes a device from the system device table (de-initializes the device).
del	This is the utility that deletes the specified files.
deldir	This is the utility that deletes the specified data directory along with the files and subdirectories contained within it.
delmdir	This is the utility that deletes existing module directories.
devs	This is the utility that displays a list of all of the initialized devices in the system.
dhcp	This is the DHCP client negotiation utility.
dir	This is the utility that displays a formatted list of file names from the specified directory.
diskcache	This is the utility that enables, disables, or displays the status of a disk cache.
DPSplit	This is the utility that is used to split and rejoin the DPIO descriptor.
dsave	This is the utility that copies a directory and its contents to another location.
dump	This produces a formatted display of the physical data contents of a mass storage file.

Table A-6. Bootfile Image Modules (Continued)

Module, Device Driver, File Manager, or Descriptor	Description
echo	This is the utility that echoes its parameter to the standard output path.
EditMod	This is the utility that creates, displays, and edits modules.
edt	This is the utility that is a line-oriented text editor that allows you to create and edit source files.
enet	This is the descriptor for the generic ethernet driver (spenet)
events	This is the utility that displays a list of the active events on the system and information about each event.
exbin	This is the utility that converts S-record files to binary.
expand	This is the utility that restores compressed files to their original form. It is the complement command of the compress utility.
exportfs	This is the utility that indicates to the NFS server system which devices can be mounted by remote hosts.
fcopy	This is a MAUI demonstration program that blits (copies) bitmap graphics to the screen.
fdisk	This is the utility that makes RBF (Random Block File Manager) disk partitions (not required for PCF IDE PC Cards).
fdraw	This is a MAUI demonstration program that draws squares to the screen.
fixmod	This is the utility that verifies and updates module parity and module CRC.
format	This is the utility that initializes the RBF (Random Block File Manager) file structure on a disk device (not required for PCF IDE PC Cards).
fpuexcpt	This is the SH-4 FPU exception handler.
free	This is the utility that displays free space remaining on a mass-storage device
frestore	This is the utility that restores a directory structure from multiple volumes of tape or disk media.
fsave	This is the utility that performs an incremental backup of a directory structure to tape(s) or disk(s).
ftp	This is the utility that contains a user interface to the file transfer protocol (FTP) server deamon process.
ftpd	This is contains the incoming FTP daemon process.
ftpdc	This is the incoming communications handler for FTP.
fun.c8	This is the image module for fcopy.
gfx	This is the default MAUI graphics descriptor for the YGV618 daughter board.

Table A-6. Bootfile Image Modules (Continued)

Module, Device Driver, File Manager, or Descriptor	Description
gfx_lcd	This is an LCD screen MAUI graphics descriptor for LCD HD66820 daughter board. This item is not currently supported.
grep	This is the utility that searches the input files for lines matching an expression.
gx_hd66420	This is an LCD MAUI graphics driver module. This item is not currently supported.
gx_ygv618	This is the default MAUI graphics driver module for YGV618 daughter board.
gxdevcap	This is the utility that displays device capabilities information about each graphic device on the system.
hc1.h0	This is a PCMCIA IDE device descriptor hc1 as the startup device (/h0).
hc1fmt	This is a hard disk device descriptor (partition 1) with formatting enabled.
hcfmt	This is a hard disk device descriptor (entire disk) with formatting enabled.
hcsiofmt	This is a super I/O RBF type device, primary master entire disk, format enabled.*
hcsio1fmt	This is a super I/O RBF type device, primary master partition 1, format enabled.*
hcsio1.ho	This is a super I/O RBF type device, partition #1. *
hdlc0	This is a descriptor module for the HDLC framer driver (sphdlc)
hello	This is a MAUI demonstration program that draws text to the screen.
help	This is a module containing the help command. help displays information about a specific utility.
hlproto	This is a protoman file manager module for user-state connections.
hostname	This is the utility that displays or sets internet name of host.
idbdump	This is the utility that displays a formatted listing of the entries in the internet database.
idbgen	This is the utility that generates network database modules.
ident	This is the utility that displays module header information and the additional information that follows the header from OS-9 memory modules.
ifconfig	This is the utility that configures network interface settings.

Table A-6. Bootfile Image Modules (Continued)

Module, Device Driver, File Manager, or Descriptor	Description
inetd	This is the module for the master internet daemon process.
inetdb	This is a LAN configuration data module.
inetdb2	This is a LAN configuration data module.
iniz	This is the utility that initializes and links the device to the system.
inp	This is a MAUI input demonstration program.
ioman	This file manager handles all I/O requests.
ip0	This is a descriptor module for the SPF IP protocol driver (spip).
ipcp0	This is a descriptor module for a PPP client driver (spipcp).
ipstart	This module initializes the IP stack.
irqs	This is the utility that displays a list of the system's IRQ polling table.
jview	This is a MAUI demo application that displays JPEG images.
kermit	This utility is an OS-9 implementation of the kermit protocol.
kernel	This is the OS-9 kernel.
kx0	This is the keyboard descriptor.
link	This is the utility that increases the link count of the specified memory module.
list	This is the utility that displays text lines from the specified path or paths (typically a file or files) to standard output.
llcis	This is a low-level PCMCIA configuration information service module.
ln	This is the utility that creates a directory entry (a hard link) that refers to a file.
load	This is the utility that loads one or more specified modules into memory.
login	This is the utility that provides login security in multi-user systems.
lcp0	This is a descriptor module for a LAN PPP client driver (splcp).
m0	This is the mouse descriptor (PS/2).
m0_t3	This is the mouse descriptor (serial).
makdir	This is the utility that creates a new directory.
make	This is the utility that rebuilds a file if any of its sources have been updated.

Table A-6. Bootfile Image Modules (Continued)

Module, Device Driver, File Manager, or Descriptor	Description
makmdir	This is the utility that creates a new module directory.
maui	This is a shared library module (contains the MAUI API).
maui_inp	This is an input daemon for MAUI applications.
maui_win	This is a window daemon for MAUI applications.
mbinstall_csl	This is the utility that installs the user-installed system call and allocates memory for use as the system mbuf pool.
mdattr	This is the utility that changes the security (access) permissions of a module directory.
mdir	This is a module containing the `mdir` basic system command. `mdir` displays the present module names in the module directory.
merge	This is the utility that copies the specified multiple input files to standard output.
mfm	This is the MAUI file manager.
mfree	This is a module containing the `mfree` basic system command. `mfree` displays a list of areas in memory not presently in use and available for assignment.
mhc1	This is a device descriptor for PCMCIA IDE drive 0, partition 1 (for socket 0).
mhc1.h0	This descriptor acts as a startup device (/h0).
mount	This is the utility that indicates to OS-9 that a file system is to be associated with a local device and accessed via NFS.
mountd	This is the daemon that answers file system mount requests.
mp_bsptr	This is a bus mouse serial protocol module.
mp_kybrd	This is a MAUI Input Process Protocol Module for generic VT100 type keyboard input.
mp_msptr	This is a MAUI Input Process Protocol Module for a two-button serial mouse.
mp_xtkbd	This is an XT scan code keyboard protocol module.
msgrdr	This is a message reading MAUI demonstration program.
msgwrtr	This is a message writing MAUI demonstration program.
mshell	This acts as an expanded command interpreter.
mt_maui	This is a shared library module (contains the MAUI API).
mv	This is the utility that moves a file or directory from one directory into another.
mwlogo.c8	This is an image module for `fcopy`.

Table A-6. Bootfile Image Modules (Continued)

Module, Device Driver, File Manager, or Descriptor	Description
ndbmod	This is the utility that is used to dynamically update the internet data module.
ndpio	This is a user-state remote debugger module for use with Hawk.
netdb_dns	This is a LAN trap handler module for DNS name resolution.
netstat	This is the utility that reports network information.
nfs	This is the NFS file manager.
nfs_devices	This is an NFS device descriptor.
nfsc	This is an NFS client auxiliary process.
nfsd	This is an NFS daemon.
nfsnul	This is an NFS device driver.
nfsstat	This is the utility that displays statistics about NFS and RPC.
nil	This is a device descriptor.
null	This is a device driver.
on	This is the utility that is used to execute a remote command.
p0	This is a printer descriptor.
p2init	This utility installs OS-9 P2 modules.
park	This is the utility that parks hard drive heads.
pcf	This is the PC File manager (MS-DOS devices)
pcmcia	This is the PCMCIA (PC Card) socket control manager command that initializes the PCMCIA socket.
pd	This is the utility that shows the path from the root directory to the current data directory.
pflash	This is the utility that clears and programs flash memory on the target.
pflashcore	This is a data module that contains a coreboot image. It is used when PF-CORE is selected in the Wizard.
pflashrom	This is a data module that contains a ROM image. Used when PF-ROM is selected in the Wizard.
ping	This module sends an ICMP echo request to a specified host and waits for a reply.
ping6	This module sends an ICMPv6 echo request to a specified host and waits for a reply.
pipe	This is a pipe descriptor.
pipeman	This is the file manager for pipes.
pjruntime	This is Microware's PersonalJava Solution runtime (available only after Microware's PersonalJava Solution has been installed).

Table A-6. Bootfile Image Modules (Continued)

Module, Device Driver, File Manager, or Descriptor	Description
`pk`	This is a module containing the descriptor for the pseudo keyboard.
`pkdvr`	This is a module containing the pseudo keyboard driver.
`pkman`	This is a module containing the file manager for the pseudo keyboard.
`portmap`	This is the daemon that converts RPC program numbers into DARPA protocol port numbers.
`pppauth`	This is the utility that creates an authentication module for `splcp`.
`pppd`	This is a module containing the PPP daemon.
`pr`	This is the utility that produces a formatted listing of one or more files to standard output.
`printenv`	This is the utility that prints any defined environment variables to standard output.
`procs`	This shows the current process list.
`pwrext`	This is the power management extension module.
`pwrman`	This is a power management module.
`pwrplcy`	This is a power management module (contains platform specific code).
`qsort`	This is the utility that performs a quicksort on any number of lines up to the maximum capacity of memory.
`r0`	This is a device descriptor for the Random Block File (RBF) RAM disk.
`r0.dd`	This is a device descriptor for the Random Block File (RBF) RAM disk as the default device.
`ram`	This is a device driver for a RAM disk.
`raw0`	This is a descriptor for the SPF RAW protocol driver (`spraw`).
`rb1003`	This is a device driver for PCMCIA IDE hard drives.
`rb1003sio`	This is a device driver for Super I/O IDE hard drives.
`rbf`	This is the Random Block File (RBF) manager (OS-9 file system devices).
`rename`	This is the utility that assigns a new name to the mass storage file specified in the pathlist.
`RomBug`	This is the RomBug debugger client module.
`romsplit`	This is the utility that splits the specified input file into two or four files.
`route`	This updates and prints the current routing table.
`route0`	This is a descriptor module for the SPF routing domain protocol driver (`sproute`)

Table A-6. Bootfile Image Modules (Continued)

Module, Device Driver, File Manager, or Descriptor	Description
`route6d`	The route6d is a routing daemon; it supports RIP over IPv6.
`routed`	This is a network routing daemon used to maintain routing tables.
`rpcdb`	This is an NFS/RPC database module.
`rpcdbgen`	This is the utility that generates an OS-9 data module from host information supplied in the `rpcdbgen` call.
`rpcdump`	This is the utility that displays information in the RPC database module `rpcdb`.
`rpcgen`	This is the utility that generates source code for implementing an RPC application.
`rpcinfo`	This is the utility that calls an RPC server in an attempt to find a single version or all versions of a specific program.
`rpr`	This copies a file to the remote system and prints it.
`rstatd`	This daemon returns statistics obtained from the kernel. `rstatd` is called by `rup`.
`rtc7750`	This is the real-time clock module.
`rtsol`	Utility to Send ICMPv6 Router Solicitation message to the specified interfaces
`rup`	This displays a system status for the specified host.
`rusers`	This displays a list of users logged into the specified host.
`rusersd`	This returns a list of users on the system.
`save`	This is the utility that copies the specified module(s) from memory into the current data directory as files.
`sc8042k`	This is a PS/2 keyboard and mouse driver.
`sc16550`	This is a serial driver for the 16550 serial ports.
`sc7750`	This is a serial driver for the SH7750 internal UART.
`scf`	This is the file manager for Sequential Character File (SCF) devices.
`scp87303`	This is a printer driver.
`scscish4`	This is a serial driver for the sci 7750 internal UART.
`setime`	This is a module containing the `setime` basic system command. `setime` sets the system date and time.
`sfont`	This is a MAUI demo application that prints information about a specified UCM font in memory.
`shell`	This is a command interpreter.
`showimg`	This is a MAUI demo application that displays a specified IFF image file for about 10 seconds.

Table A-6. Bootfile Image Modules (Continued)

Module, Device Driver, File Manager, or Descriptor	Description
`showmount`	This is the utility that displays the remote hosts and the local OS-9 devices mounted to the OS-9 NFS server.
`sleep`	This is the utility that puts a running process to sleep for a specified amount of time.
`sndp`	This is a system-state debugging client.
`sp7750`	This is a driver for NS DP835902 Ethernet controller.
`spne0`	This is a descriptor for the Ethernet driver `sp7750`.
`spenet`	This is a generic ethernet driver module.
`spf`	This is a module containing the SPF (Stackable Protocol File) manager.
`spfndpd`	This is a user-state remote debugger module (network debugger protocol server daemon).
`spfndpdc`	This is a user-state remote debugger module (network debugger protocol server connection handler).
`spfnppd`	This is the Hawk Profiler server daemon module.
`spfnppdc`	This is the Hawk Profiler server connection handler.
`sphdlc`	This is a HDLC framer driver module that is part of the PPP stack.
`spip`	This is a module containing the SPF IP protocol driver.
`spipcp`	This is the module that implements the Network Control Protocol for IP with the PPP stack.
`splcp`	This is the module that implements the LCP protocol within the PPP stack.
`spraw`	This is a module containing the SPF RAW protocol driver (standard raw socket interface into the IP layer).
`spray`	This is sends a one-way stream of packets to the host using RPC and reports how many were received by the host and the transfer rate.
`sprayd`	This records the packets sent by the `spray` RPC client.
`sproute`	This is a module containing the SPF routing domain protocol driver
`spsl0`	This is a module containing the descriptor for the SLIP protocol driver (`spslip`)
`spslip`	This is a module containing the SLIP protocol driver
`sptcp`	This is a module containing the SPF TCP protocol driver
`spudp`	This is a module containing the SPF UDP protocol driver
`ssm`	This is a MMU module that provides processes with address space protection
`su`	This is the utility that allows you to start a new shell with a different user ID.

Table A-6. Bootfile Image Modules (Continued)

Module, Device Driver, File Manager, or Descriptor	Description
`suspend`	This is the utility that de-activates or suspends an active process.
`sysid`	This is the utility that prints out the identification information of the system.
`sysif`	This is a power management module that provides a system specific interface to hardware components that do not have a device driver interface to OS-9.
`sysmbuf`	This is a module that controls the allocation and deallocation of mbufs from the system mbuf free pool.
`t1`	This is a device descriptor for the SCIF port of the 7750 internal UART.
`t1_auto`	This is a device descriptor for the SCIF port of the 7750 internal UART (automatic CTS/RTS).
`t1_hw`	This is a device descriptor for the SCIF port of the 7750 internal UART1 (hardware flow control).
`t2`	This is a device descriptor for the SCI port of the 7750 internal UART.
`t3`	This is a device descriptor for the 16550-compatible port 1.
`t4`	This is a device descriptor for the 16550-compatible port 2.
`tape`	This is the utility that provides a means to access a tape controller from a terminal.
`tapegen`	This is the utility that creates a "bootable" tape.
`tar`	This is the utility that archives multiple files or directories onto a magnetic tape or file.
`tcp0`	This is a descriptor for the SPF TCP protocol driver (`sptcp`).
`tee`	This utility is a filter that copies all text lines from its standard input to its standard output as well as any specified path lists.
`telnet`	This is the utility that allows the user to execute commands on a remote host.
`telnetd`	This is the telnet server daemon process.
`telnetdc`	This is the telnet server connection handler.
`term1`	This is a device descriptor for using the console through the 7750 SCIF port.
`term1_auto`	This is a device descriptor for using the console through the 7750 SCIF port (automatic CTS/RTS).
`term1_hw`	This is a device descriptor for using the console through the 7750 SCIF port (hardware flow control).

Table A-6. Bootfile Image Modules (Continued)

Module, Device Driver, File Manager, or Descriptor	Description
term2	This is a device descriptor for using the console through the 7750 SCI port.
term3	This is a device descriptor for using the console through the 16550-compatible port 1.
term4	This is a device descriptor for using the console through the 16550-compatible port 2.
tftpd	This is the TFTP Server Daemon.
tftpdc	This is the TFTP Server Connection Handler.
tk7750	This is a system clock module.
tmode	This is a module containing the tmode basic system command. tmode displays or changes the operating parameters for an I/O path.
touch	This is a utility that updates the last modification date of a file.
tr	This is a utility that transliterates characters from string 1 into a corresponding character from string 2.
transh4	This is the address translation module for SH7750.
travel.c8	This is an image module for fdraw
tsmon	This is the utility that supervises idle terminals and starts the login utility in a timesharing application.
udp0	This is a descriptor module for the SPF UDP protocol driver (spudp).
umacs	This is a screen-oriented text editor used to create and modify text files.
undel	This utility allows you to copy the data of the deleted file to a new file on another device.
undpd	This is a low-level user-state remote debugger module (network debugger protocol server daemon)
undpdc	This is a low-level user-state remote debugger module (network debugger protocol server connection handler)
unlink	This utility reduces the specified modules link count by one. When the link count reaches zero, OS-9 will remove the module from memory.
vectsh7750	This is a vector module for SH7750.
windraw	This is a window based block drawing MAUI demonstration program.
winink	This is a window based pen drawing MAUI demonstration program.
winmgr	This is the MAUI demo Window Manager.
xmode	This utility displays or changes the default operating parameters for a device.