Home

OS-9® for IDT 79S334 Board Guide

Version 4.7

RadiSys
THE POWER OF WE

www.radisys.com

Revision A • July 2006

Copyright and publication information

This manual reflects version 4.7 of Microware OS-9.
Reproduction of this document, in part or whole, by any means, electrical, mechanical, magnetic, optical, chemical, manual, or otherwise is prohibited, without written permission from RadiSys Microware Communications Software Division, Inc.

Disclaimer

The information contained herein is believed to be accurate as of the date of publication. However, RadiSys Corporation will not be liable for any damages including indirect or consequential, from use of the OS-9 operating system, Microware-provided software, or reliance on the accuracy of this documentation. The information contained herein is subject to change without notice.

Reproduction notice

The software described in this document is intended to be used on a single computer system. RadiSys Corporation expressly prohibits any reproduction of the software on tape, disk, or any other medium except for backup purposes. Distribution of this software, in part or whole, to any other party or on any other system may constitute copyright infringements and misappropriation of trade secrets and confidential processes which are the property of RadiSys Corporation and/or other parties. Unauthorized distribution of software may cause damages far in excess of the value of the copies involved.

Contents

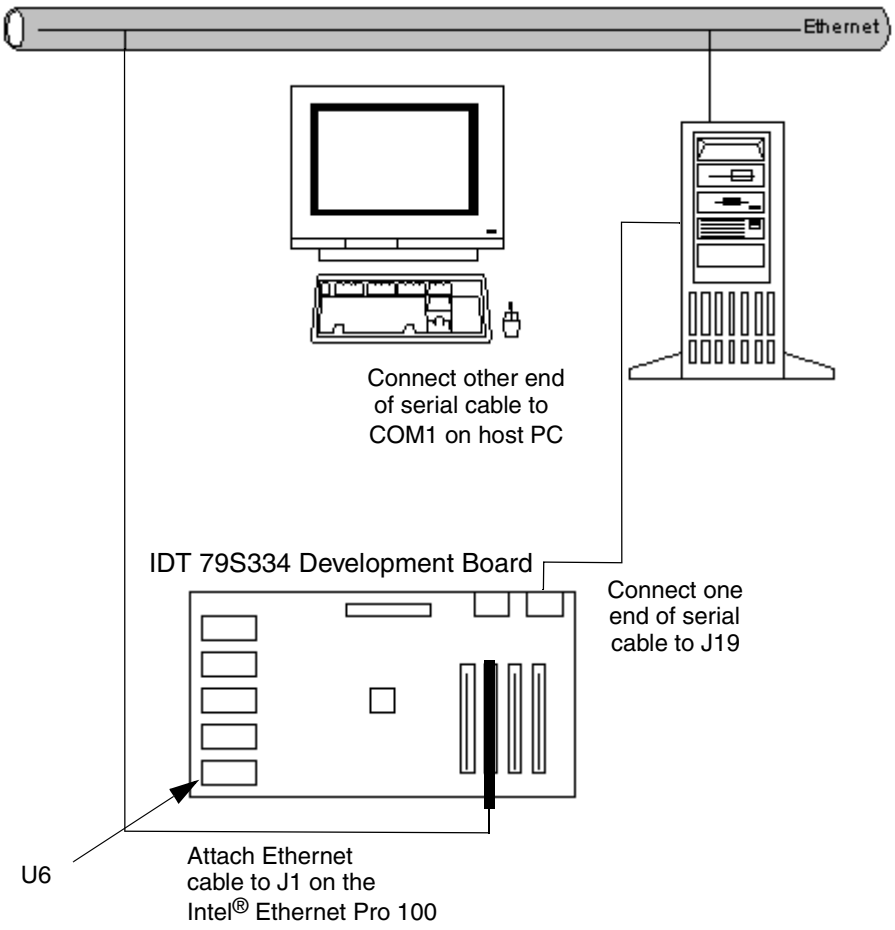
1 Installing and Configuring OS-9®

This chapter describes installing and configuring OS-9® on the MIPS IDT 79S334 board. It includes the following sections:

Development Environment Overview

Figure 1-1 shows a typical development environment for the MIPS board. The components shown include the minimum required to enable OS-9 to run on the MIPS IDT 79S334 board.

Figure 1-1. IDT79S334 Development Environment



Requirements and Compatibility

Before you begin, install the *Microware 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 (for down-loading programs to the board)

Target and Host Setup

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

Jumper and Switch Settings

The factory default setting for the DIP switches will work with OS-9. If you need to change any of the jumper or switch settings, please refer to the *IDT79S334A Evaluation Board Manual*.

Installing the TFTP Server

This section details the steps involved with setting up the Walusoft TFTP server on your host machine. If you choose to use another TFTP server in place of the Walusoft, be certain to follow its directions and specifications.

To set up the Walusoft TFTP server on your host machine, complete the following steps:

1. Load the product CD into the host machine's CD ROM drive and let the CD autorun. The installer's dialog box appears.
2. Select `Walusoft TFTPServer32Pro` from the installer's menu and follow the installer's directions to load the TFTP server on the host machine.
3. Start TFTPServer32Pro by selecting `Start` -> `Programs` -> `TFTPServer` -> `TFTPServer32`. The Walusoft TFTP Server32 Pro splash screen appears. You will need to set up the path to the outbound folder.
4. Select `System` -> `Setup` to display the Server Options dialog box.
5. Click on the Outbound tab and enter the following path in the Outbound file path text box.
   `<drive>:\mwos\OS9000\MIPS32\PORTS\IDT_79S334\BOOTS\INSTALL\PORTBOOT\`

6. Click OK to accept the path. The TFTP server is now configured for use with the Configuration Wizard and OS-9.
7. Minimize the TFTP server window to let the server run in the background.

Connecting the Target to the Host

Connecting the IDT 79S334 to your host PC involves attaching the power, serial, and Ethernet cables to the reference board. Once you have the board connected, you can use the serial console in Hawk™ to verify the serial connection.

Attaching the Cables

1. Attach an Ethernet cable to connector J1 on the Intel® Ethernet Pro 100 card.
2. Connect a serial cable to connector J19 on the IDT 79S334 board for the serial console.
3. Connect the other end of the serial cable to COM1 on the host PC. Depending on your PC system, you may need either a straight or a reversed serial cable to make this connection.

A standard null modem cable may not work. You may need a cable that has transmit and receive swapped.

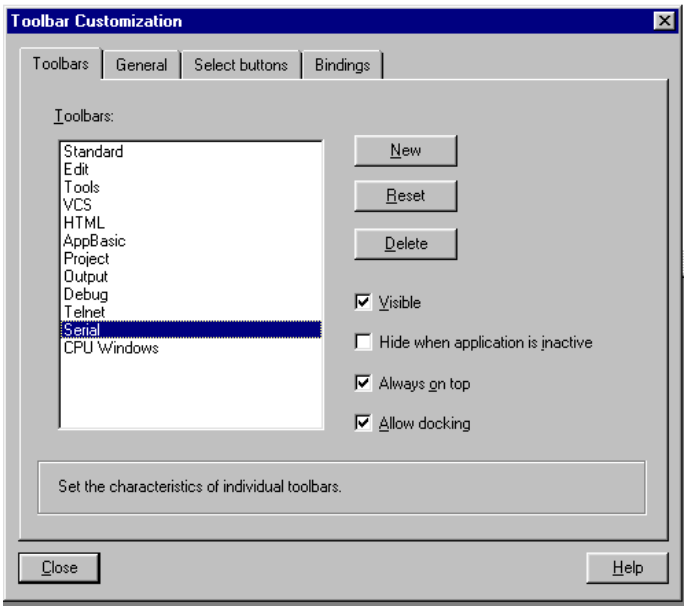
4. Following the instructions in the *IDT 79S334 Evaluation Board Manual*, attach a PC power supply.

Booting to the Boot Menu

You may want to boot to the IDT System Integration Manager prompt to verify that your serial cable is connected properly.

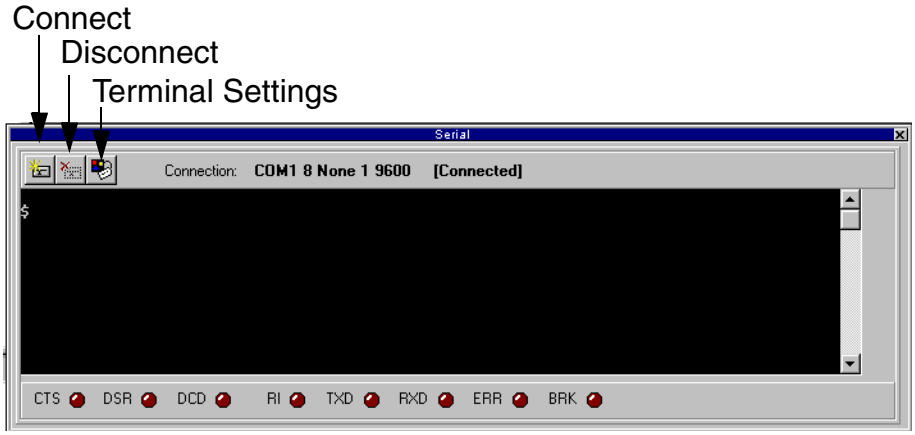
1. From the desktop, click `Start` and select `Programs -> Microware -> Microware OS-9 for MIPS v3.0 -> Microware Hawk IDE` to start the Microware Hawk IDE.
2. If the Serial console window is not open, it can be opened from the Toolbar Customization dialog (shown in Figure 1-2). (Select `Tools -> Customize -> Toolbars` to open the Toolbar Customization dialog.)

Figure 1-2. Toolbar Customization dialog box



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

Figure 1-3. Hawk Serial Console Window



5. Once you have the serial console window open, click on the `Connect` button in the upper left corner of the serial console window. The Com Port Options dialog box appears.
6. Click on the `OK` button because the default settings are correct. The message *[Not Connected]* should change to *[Connected]*.
7. Apply power to the board. The IDT System Integration Manager boots the board. The display looks similar to the following figure.

Figure 1-4. IDT SIM initial screen

```
Please Wait. Scanning Memory ...
PCI slot 2/0: Intel 82557 (network, ethernet)
fxp0: Intel PRO/100+ Management Adapter address 00:02:b3:1c:34:30

IDT System Integration Manager Ver. 9.2, December 2000
Copyright 1994-2000 Integrated Device Technology, Inc.

RC32334 CPU, 32-bit, Big Endian, MIPS-II, Write-Back cache
Console: 9600 baud

Used for Ethernet Storage: 0xA1E24000 - 0xA1E64000
Instruction Cache: 8 KB,   Data Cache: 2 KB
Memory Configuration: SDRAM only.
Primary User Memory: 0xA019B408 to 0xA1E24000. Size: 29218 KB

CAUTION: "C" time functions such as clock() depend on the frequency of
the crystal in socket Y2. Please compare against wall-clock to obtain
 a scaling factor if needed!

           For HELP enter '?'
<IDT>
```

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.

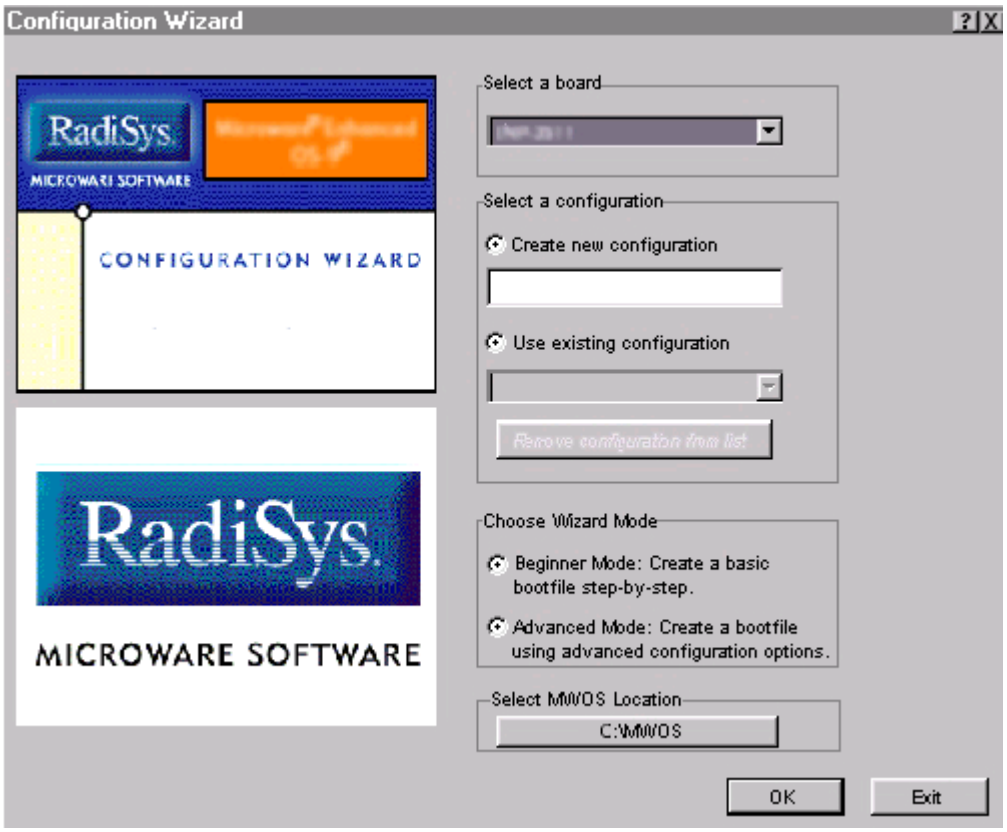
Microware provides a Configuration Wizard to create a coreboot image, a bootfile image, or an entire OS-9 ROM Image. The wizard can also be used to modify an existing image. The Configuration Wizard is automatically installed on your host PC during the OS-9 installation process.

Starting the Configuration Wizard

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

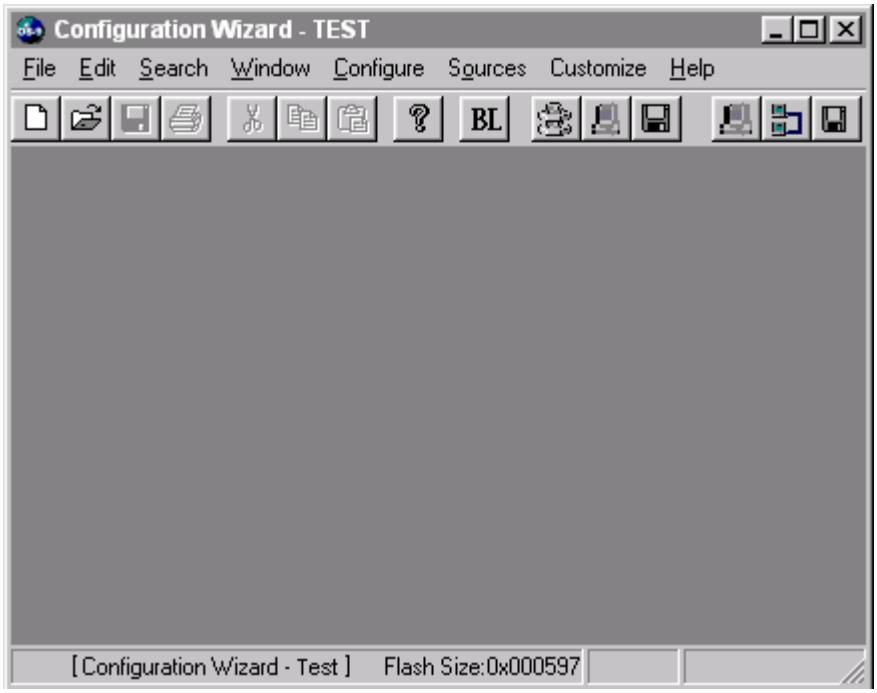
1. From the Windows desktop, select `Start` -> `RadiSys -> Microware OS-9 for <product> -> Configuration Wizard`. You should see the following opening screen:

Figure 1-5. Configuration Wizard Opening Screen



2. Select your target board from the Select a board pull-down menu.
3. Select the `Create new configuration` radio button from the Select a configuration menu and type in the name you want to give your ROM image in the supplied text box. This names your new configuration, which can later be accessed by selecting the Use existing configuration pull down menu.
4. Select the `Advanced Mode` radio button from the Choose Wizard Mode field and click `OK`. The Wizard's main window is displayed. This is the dialog from which you will proceed to build your image. An example is shown in Figure 1-6.

Figure 1-6. Configuration Wizard Main Window



Configuring Coreboot Options

Most of the default options in the dialogs that control the configuration of the bootfile are correct. There are a few functions, such as Ethernet, that need additional information in order to be configured correctly. To set up the coreboot image, complete the following steps.

1. From the menu bar, select `Configure` -> `Coreboot` -> `Main Configuration`.
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.
3. Click on 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 text boxes:
   - IP Address
   - IP Broadcast
   - Subnet Mask
   - IP Gateway

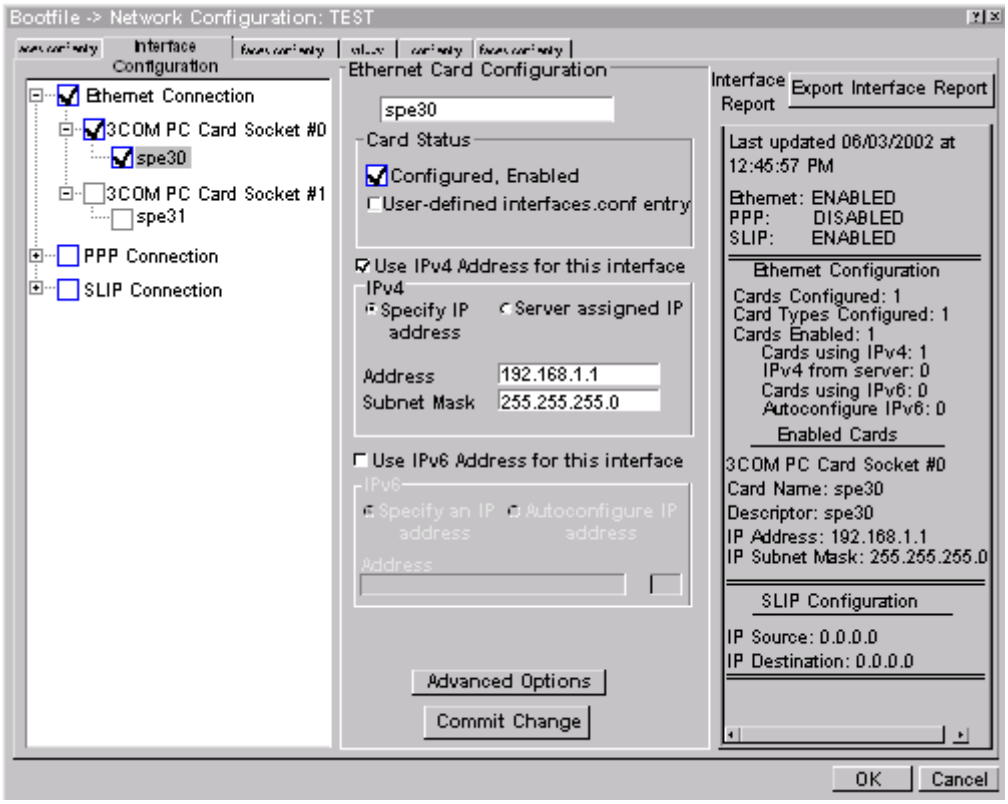
   If you are uncertain of the values for these text boxes, contact your system administrator.
4. Click `OK` to close the window.

Configuring Bootfile Options

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

1. Select `Configure -> Bootfile -> Network Configuration` from the Wizard's main menu.
2. From the Network Configuration dialog, select the `Interface Configuration` tab. From here you can select and enable the interface. For example, you can select the appropriate Ethernet card from the list of options on the left and specify whether you would like to enable IPv4 or IPv6 addressing. Figure 1-7 shows an example of the Interface Configuration tab.

Figure 1-7. Bootfile -> Network Configuration -> Interface Configuration



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

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

3. Click on the SoftStax® Setup tab, and select Enable SoftStax.
4. Click OK to close the dialog box.
5. Select Configure -> Bootfile -> Disk Configuration.. from the menu bar and verify that the default settings in the dialog box are acceptable to you.
6. Leave the other default settings alone and select Configure -> Build Image.. from the menu bar to display the Master Builder window.
7. Select the following check boxes as they are appropriate to your setup:

- SoftStax (SPF) Support
- User State Debugging Modules
- If you are using a RAM disk, select Disk Support.
- If you are using a RAM disk, select Disk Utilities.

1. Click Coreboot + Bootfile and click Build. This will build 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 mwos\OS9000\MIPS32\PORTS\IDT_79S334\BOOTS\INSTALL\PORTBOOT. This directory was specified as the outgoing directory when the TFTP server was set up.
2. Click Finish and then select File -> Save Settings to save the configuration.
3. Select File -> Exit to quit from the Configuration Wizard.

Transferring the ROM Image to the Target

In the previous section, you built a ROM image. To load this ROM image onto the target board, complete the following steps:

1. The networking environment variables on the IDT board needs to be set up before you use it for the first time. The ethaddr environment variable needs to be set with the setenv command. To set these variables, type the following commands at the IDT SIM (System Integration Manager) prompt:

   ```
   setenv netaddr <IDT board's IP address>
   ```

   Download the OS-9 ROM image, rom.S, to the IDT board using the following load command at the IDT SIM (System Integration Manager) prompt:

   ```
   l -t -s <Host Machine's IP address>:rom.S
   ```

It will take six or seven minutes to download the entire image.

If you have trouble with downloading the image, be sure the following items are correct:

- The environment variables on the board are set to correct values.
- The load command was correctly entered (use of spaces is correct).
- The path to the Outbound folder in your TFTP server is correct.
- The Ethernet connector is plugged in and the board has power applied to it.

2. Enter the following command to start OS-9.
   ```
   go 80200000
   ```
3. To be able to use Hawk to load and debug your applications, you need to start the debugging daemons. Type the following command to start the debugging daemons:
   ```
   spfndpd<>>>/nil&
   ```

Optional Procedures

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

Building with Makefiles

Building boots with makefiles allows you greater control over which modules are included in the boot. For the IDT79S334 reference board, there are two directories in which boots can be made. These are shown below:

`MWOS\OS9000\MIPS32\PORTS\IDT_79S334\BOOTS\SYSTEMS\PORTBOOT`

In the above directory, boots can be made that use the IDT SIM (System Integration Manager) to load the OS-9 ROM image.

`MWOS\OS9000\MIPS32\PORTS\IDT_79S334\BOOTS\SYSTEMS\EPROMBOOT`

In the above directory, boots can be made that can be burned into the flash parts on the 79S334 reference board. The boots in the flash parts replace the IDT SIM.

EPROMCORE vs. PORTBOOT

The difference between the two boots explained in the previous section is the memory lists. When using the IDT SIM to boot OS-9, part of the RAM must be allocated as ROM, where the bootfile is located. When not using the IDT SIM (instead using OS-9 in the flash), the full RAM can be used for system memory. This also gives a wider range of booting options.

The memory list difference not only shows up in a different romcore, but also in the different init modules as well. Hence there is a separate set of files for EPROMCORE in the subsequent EPROM directories. There is also the `EPROM` macro definition and condition that selects the appropriate memory list for `romcore` and the init modules. This `EPROM` conditional is set in the `systype.h` file and the `INIT\default.des` file.

Makefile Network Option

By default the makefile in the `EPROMCORE` and `PORTBOOT` directories will not include networking. However, by setting the network macro definition to `TRUE`, the networking modules will be included in the bootfile. In addition, be sure the IP and MAC addresses for the board are setup correctly to avoid network problems.

Because of the size of the IDT_79S334 EPROM, it is not possible to put SPF into the EPROM. So builds out of the EPROMCORE directory should not include SPF.

Using Makefiles

When using a makefile to build boots, three bootlist files are used to include the modules for booting. These bootlist files can be edited in order to include or not include modules needed for your system. These bootlist files are located in `IDT_79S334\BOOTS\SYSTEMS\PORTBOOT` and `IDT_79S334\BOOTS\SYSTEMS\EPROMBOOT` They are defined as follows:

`coreboot.ml` — used to make the low-level boot (called `coreboot`)

When using this file, the `romcore` file must be input first, followed by the `initext` file. These two files are not OS-9 modules. `romcore` is the raw code needed to bring the hardware to a known stable state, while `initext` is a way for users to extend the low level `sysinit` code without changing `sysinit.c` or remaking `romcore`.

The rest of the files included with `coreboot.ml` are actual OS-9 modules. Low-level booters and debuggers can be added or removed. In addition, the low-level Ethernet and IP stack can be uncommented in order to provide `bootp` booting. Low-level Ethernet or low-level SLIP can also provide system state debugging through Hawk.

Only OEM licensees have the ability to make romcore. BLS licensees do not have this ability.

`bootfile.ml` — used to create the high-level boot (called `bootfile`)

This file contains all of the modules needed to produce an OS-9 system. This includes the kernel, system protection, cache control, file managers, and drivers and descriptors. Also included are various utilities and application programs.

Not included with this file are networking modules. Additional modules can be included or excluded where appropriate.

`spf_mods.ml` — contains the SoftStax modules and network utilities

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

Making Network Configuration Changes

To configure the network parameters for SoftStax and Ethernet, two files need to be changed and two makefiles need to be run. To do this, complete the following steps:

1. Navigate to `MWOS\OS9000\MIPS32\PORTS\IDT_79S334\SPF\ETC` directory and open the `interfaces.conf` file.
2. From the `interfaces.conf` file, fill in the correct IP address, broadcast address, and netmask values. You can also supply the host name in this area as well.
3. Save the file.
4. Once you have saved the file, run the makefile in the directory listed in step one. This will make the appropriate `inetdb` and `inetdb2` modules.
5. Once this file is saved, go to the `MWOS\OS9000\MIPS32\PORTS\IDT_79S334\SPF\SPPRO100` directory and run the `spfdesc.mak` makefile. This will create the `spse0` descriptor. At this point networking is configured for SoftStax.

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

Low Level Network Configuration Changes

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

1. Navigate to the `MWOS\OS9000\MIPS32\PORTS\IDT_79S334\ROM\CNFGDATA` directory and open the `config.des` file.
2. In the `config.des` file, you will need to correctly define the macros for the IP, broadcast, subnet, and MAC addresses.
3. Run the makefile in the directory listed in step one and a new `cnfgdata` module will be created. A coreboot can now be created with this configuration.

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

2 Board Specific Reference

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

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 you a considerable amount of time. For example, if a system has many modules in ROM, in which access time is typically longer than it is in RAM, omitting the CRC check will drastically decrease the bootstrap time. Furthermore, since it is rare that data corruption will occur in ROM, omitting the CRC check is a safe option.

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

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

B_OKROM

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

B_1STINIT

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

B_NOIRQMASK

The `B_NOIRQMASK` bit instructs the entire bootstrap system to not mask interrupts for the duration of the bootstrap process. Normally, the ModRom code and the kernel cold-start mask interrupts for the duration of the system startup. However, in systems with a well-defined interrupt system (systems that are calmed by the `sysinit` hardware initialization code) and a requirement to respond to an installed interrupt handler during startup, this option can be used. Its implementation will prevent the ModRom and kernel cold-start from disabling interrupts. (This is useful in power-sensitive systems that need to respond to "power-failure" oriented interrupts.)

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

B_NOPARITY

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

Implementation Details

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

Compile-time Configuration

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

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

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

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

OS-9 Vector Mappings

This section contains the vector mappings for the IDT79S334.

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

Table 2-1. IRQ Assignments

OS9 IRQ #	MIPS Function
0xd0	Bus error
0x40	Programable I/O pin 0 is low
0x41	Programable I/O pin 1 is low
0x42	Reserved
0x43	Programable I/O pin 2 is low
0x44	Programable I/O pin 3 is low
0x45	Programable I/O pin 4 is low
0x46	Programable I/O pin 5 is low
0x47	Programable I/O pin 6 is low
0x48	Programable I/O pin 7 is low
0x49	Programable I/O pin 8 is low
0x4A	Programable I/O pin 9 is low
0x4B	Programable I/O pin 10 is low
0x50	Programable I/O pin 0 is high
0x51	Programable I/O pin 1 is high
0x52	Reserved
0x53	Programable I/O pin 2 is high
0x54	Programable I/O pin 3 is high
0x55	Programable I/O pin 4 is high

Table 2-1. IRQ Assignments (Continued)

OS9 IRQ #	MIPS Function
0x56	Programable I/O pin 5 is high
0x57	Programable I/O pin 6 is high
0x60	Timer 0 rollover
0x61	Timer 1 rollover
0x62	Timer 2 rollover
0x63	Watchdog/timer 3 rollover
0x64	CPU bus timout/timer 4 rollover
0x65	IP bus timout/timer 5 rollover
0x66	DRAM refresh/timer 6 rollover
0x67	ColdReset/timer 7 rollover
0x70	UART0 IIR interrupt
0x71	UART0 TxRdy interrupt
0x72	UART0 RxRdy interrupt
0x78	UART1 IIR interrupt
0x79	UART1 TxRdy interrupt
0x7a	UART1 RxRdy interrupt
0x80	DMA channel 0 done
0x81	DMA channel 0 end too early error
0x82	DMA channel 0 descriptor not owned error
0x83	DMA channel 0 transaction complete
0x84	DMA channel 0 clear
0x88	DMA channel 1 done

Table 2-1. IRQ Assignments (Continued)

OS9 IRQ #	MIPS Function
0x89	DMA channel 1 end too early error
0x8a	DMA channel 1 descriptor not owned error
0x8b	DMA channel 1 transaction complete
0x8c	DMA channel 1 clear
0x90	DMA channel 2 done
0x91	DMA channel 2 end too early error
0x92	DMA channel 2 descriptor not owned error
0x93	DMA channel 2 transaction complete
0x94	DMA channel 2 clear
0x98	DMA channel 3 done
0x99	DMA channel 3 end too early error
0x9a	DMA channel 3 descriptor not owned error
0x9b	DMA channel 3 transaction complete
0x9c	DMA channel 3 clear
0xa0	PCI master write error
0xa1	PCI master read error
0xa2	PCI master data parity error
0xa3	PCI target write data parity error
0xb0	PCI UART0 IIR interrupt
0xb1	PCI UART0 TxRdy interrupt
0xb2	PCI UART0 RxRdy interrupt
0xb3	PCI UART1 IIR interrupt

Table 2-1. IRQ Assignments (Continued)

OS9 IRQ #	MIPS Function
0xb4	PCI UART1 TxRdy interrupt
0xb5	PCI UART1 RxRdy interrupt
0xb6	PCI DMA MEM2IO 0 done
0xb7	PCI DMA MEM2IO 0 end too early error
0xb8	PCI DMA MEM2IO 0 descriptor not owned error
0xb9	PCI DMA MEM2IO 1 done
0xba	PCI DMA MEM2IO 1 end too early error
0xbb	PCI DMA MEM2IO 1 descriptor not owned error
0xbc	CPU2PCI mailbox interrupt 0
0xbd	CPU2PCI mailbox interrupt 1
0xbe	CPU2PCI mailbox interrupt 2
0xbf	CPU2PCI mailbox interrupt 3
0xc0	PCI2CPU mailbox interrupt 0
0xc1	PCI2CPU mailbox interrupt 1
0xc2	PCI2CPU mailbox interrupt 2
0xc3	PCI2CPU mailbox interrupt 3
0xc8	Serial peripheral interface

A Board Specific Modules

This chapter describes the modules specifically written for the MIPS 79S334 boards. It includes the following sections:

<Bold><links>Low-Level System Modules

<Bold><links>High-Level System Modules

Low-Level System Modules

The following low-level system modules are tailored specifically for the IDT 79S334 board. They are located in the following directory:

`MWOS/OS9000/MIPS32/PORTS/IDT_79S334/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 IO driver
`llpro100`	Low-level Ethernet ROM driver
`portmenu`	inits booters defined in the `cnfgdata`
`romcore`	bootstrap code
`tmr32334`	ROM timer services
`usedebug`	debugger configuration module

High-Level System Modules

The following OS-9 system modules are tailored specifically for the MIPS IDP 79S334 boards. Unless otherwise specified, each module is located in the following directory:

`MWOS/OS9000/MIPS32/PORTS/IDT_79S334/CMDS/BOOTOBJS`

`counter`	Dummy IRQ handler that handles MIPS 32 counter interrupts
`sc16550`	Serial driver for the 16550 UART
`pcirq`	Provide interrupt acknowledge and dispatching support for the 32334 on-chip interrupt controller.
`tk32334`	System clock module
`vectmips32`	Vector module for MIPS 32

Common System Modules List

The following low-level system modules provide generic services for OS9000 Modular ROM. Your board port may or may not have these modules depending on your reference board's capabilities.They are located in the following directory:

`MWOS/OS9000/MIPS32/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

`fdc765`	provides PC style floppy support
`fdman`	is a target-independent booter support module providing general booting services for RBF file systems
`flboot`	is a SCSI floptical drive disk booter
`flshcach`	provides low-level cache management services
`fsboot`	is a SCSI TEAC floppy disk drive booter
`hlproto`	provides user level code access to protoman
`hsboot`	is a SCSI hard disk drive booter
`ide`	provides target-specific standard IDE support, including PCMCIA ATA PC cards
`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
`pcman`	provides a booter that reads MS-DOS file system
`protoman`	provides a protocol management module
`restart`	provides a booter that causes a soft reboot of the system
`romboot`	provides a booter that allows booting from ROM
`rombreak`	provides a booter that calls the installed debugger
`rombug`	provides a low-level system debugger
`scsiman`	is a target-independent booter support module that provides general SCSI command protocol services
`sndp`	provides low-level system debug protocol
`srecord`	provides a booter that accepts S-Records
`swtimer`	provides timer services via software loops
`tsboot`	is a SCSI TEAC tape drive booter
`type41`	is a primary partition type
`vsboot`	is a SCSI archive viper tape drive booter